



**UNIWERSYTET RZESZOWSKI**  
**Kolegium Nauk Przyrodniczych**

Jakub Gurgacz

Nr albumu: 096463

Informatyka

**ODKRYWANIE ZBIORÓW CZĘSTYCH W DANYCH STRUMIENIOWYCH – IMPLEMENTACJA  
ALGORYTMU ESTDEC**

Praca inżynierska

Praca wykonana pod kierunkiem

dra Wojciecha Rząsy

Rzeszów, 2020







**RZESZOW UNIVERSITY**

**College of Natural Sciences**

Jakub Gurgacz

096463

Computer Science

**FREQUENT ITEMSETS DISCOVERY IN DATA STREAMS;**

**IMPLEMENTATION OF ESTDEC ALGORITHM**

**Type of the thesis:** Engineer

**The thesis written under the supervision of**

dr Wojciech Rząsa

Rzeszów, 2020





Składam serdeczne podziękowania dla  
dra Wojciecha Rząsy za pomoc oraz  
wsparcie podczas wykonywania niniejszej pracy.





# Spis treści

<b>1. Wstęp .....</b>	<b>11</b>
<b>2. Modele eksploracji zbiorów częstych w strumieniach danych .....</b>	<b>12</b>
2.1. Model Landmark Window .....	13
2.2. Model Fading Window .....	14
2.3. Model Sliding Window .....	15
2.4. Model Time-tilted Window .....	16
2.5. Kompromis między dokładnością, czasem przetwarzania i wykorzystaną pamięcią .....	17
<b>3. Algorytm estDec .....</b>	<b>18</b>
3.1. Koncepcja .....	18
3.2. Założenia początkowe .....	20
3.3. Szacowanie licznika zbioru elementów .....	21
3.4. Metoda estDec .....	23
3.4.1. Faza I .....	25
3.4.2. Faza II .....	25
3.4.3. Faza III .....	25
3.4.4. Faza IV .....	27
<b>4. Symulacja działania algorytmu estDec .....</b>	<b>28</b>
<b>5. Implementacja algorytmu estDec .....</b>	<b>34</b>
5.1. Wykorzystane narzędzia i technologie .....	34
5.1.1. Java .....	34
5.1.2. JavaFX .....	34
5.1.3. JavaFX Scene Builder .....	35
5.1.4. Apache NetBeans IDE .....	35
5.1.5. Spring Boot .....	36
5.2. Struktura projektu .....	36
5.3. estDec – implementacja w Javie .....	39
5.3.1. Charakterystyka systemu .....	39
5.4. Interfejs użytkownika .....	48
5.5. Testy aplikacji .....	49
<b>6. Podsumowanie i wnioski końcowe .....</b>	<b>52</b>
<b>Literatura .....</b>	<b>55</b>
<b>Wykaz ilustracji .....</b>	<b>56</b>
<b>Streszczenie .....</b>	<b>57</b>
<b>Załączniki .....</b>	<b>58</b>



# 1. Wstęp

Postęp w technologii przechowywania danych w ostatnich latach umożliwił organizacjom przechowywanie dużych ilości danych transakcyjnych. Jednak ciągły rozwój pociąga za sobą coraz większą ilość generowanych danych.

Jednym z przykładów takiej technologii może być sieć czujników lepiej znana jako Internet of Things (IoT). Technologia ta, umożliwia monitorowanie zdarzeń w czasie rzeczywistym. W efekcie, zbieranie danych generowanych przez takie sieci, w celu dalszej analizy może nie być efektywne. Zdecydowanie korzystniejszym podejściem jest analiza tych danych w czasie rzeczywistym. Jednocześnie dane te mogą nieść ze sobą cenną wiedzę dla osób prowadzących analizę.

Na przestrzeni ostatnich lat zostały zaproponowane różne modele eksploracji zbiorów częstych w strumieniach danych. Każdy model odpowiada innemu sposobowi analizy strumienia.

Niniejsza praca prezentuje cztery główne modele analizowania danych w strumieniach danych, a także przedstawia implementację algorytmu estDec. Algorytm ten reprezentuje jeden z modeli do odkrywania ostatnich zbiorów częstych w strumieniach danych.

## 2. Modele eksploracji zbiorów częstych w strumieniach danych

Strumień danych jest nieskończoną sekwencją elementów służącą do reprezentowania danych, które są udostępniane w miarę upływu czasu. Przykładami mogą być odczyty z czujników w aplikacji do monitorowania środowiska, notowania giełdowe w aplikacjach finansowych lub dane sieciowe aplikacji do monitorowania ruchu w sieci [10].

Strumienie danych, w kontekście ich analizy, różnią się od konwencjonalnie przechowywanych modeli relacyjnych w trzech głównych aspektach [1]:

- ciągłość - dane przyrastają ciągle z dużą prędkością,
- wygaśnięcie - dane mogą być odczytane tylko raz,
- nieskończoność - liczba danych w strumieniu nie ma z góry narzuconego końca.

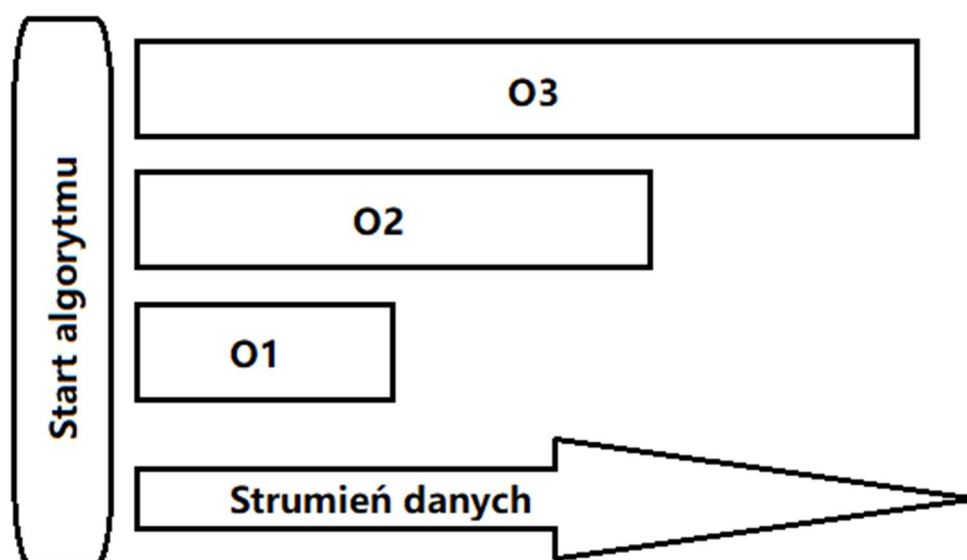
Zbiór częsty jest formą powtarzającego się wzoru. Można wskazać przykłady, które są zbiorami elementów występujących (na przykład w strumieniu danych) z minimalną częstotliwością. Każdy zbiór elementów, który występuje co najmniej w minimalnej liczbie przykładów, jest zbiorem częstym [9].

W kontekście analizy, zbiór częsty to taki zbiór, którego minimalne wsparcie jest większe od tego, które zdefiniowane zostało przez użytkownika. Istotne jest, by przechowywać nie tylko te zbiory, które są częste, ale również takie, które mogą stać się częste. W związku z tym, w wielu istniejących algorytmach stosuje się łagodny próg minimalnego wsparcia, definiowany również jako parametr błędu określonego przez użytkownika. Ma to na celu uzyskanie dodatkowego zestawu zbiorów, które mogą stać się częste w przyszłości [1].

Transakcyjny strumień danych to sekwencja transakcji przychodzących, a fragment strumienia określa się mianem „okna”. Okno może być zdefiniowane przez przedział czasu lub licznik zawartych w nim rekordów (transakcji). Jednocześnie okna można analizować przy użyciu czterech głównych modeli. Wyróżnia się model przesuwny (Sliding Window Model), oparty na punkcie orientacyjnym (Landmark Model), zapadający (Fading Window Model) lub pochylony w czasie (Time-tilted Window Model) [3].

## 2.1. Model Landmark Window

W modelu Landmark Window grupowanie rozpoczyna się od początkowego punktu (zazwyczaj jest to czas rozpoczęcia analizy strumienia), zwanego „punktem orientacyjnym”, do aktualnego punktu w czasie. Okno może być zdefiniowane jako numer obserwowanych elementów w oknie od ostatniego wybranego punktu lub jako jednostka czasu (raz na dzień, raz w tygodniu, itp.). W momencie rozpoczęcia nowego okresu okna, wszystkie dane przechowywane we wcześniejszym punkcie orientacyjnym są usuwane. Specjalnym przypadkiem w oknie z punktem orientacyjnym jest sytuacja, gdy punkt początkowy jest równy 1. W tym przypadku cała historia strumienia jest poddawana analizie. Wszystkie dane wzięte pod uwagę są traktowane przez ten model tak samo [3].



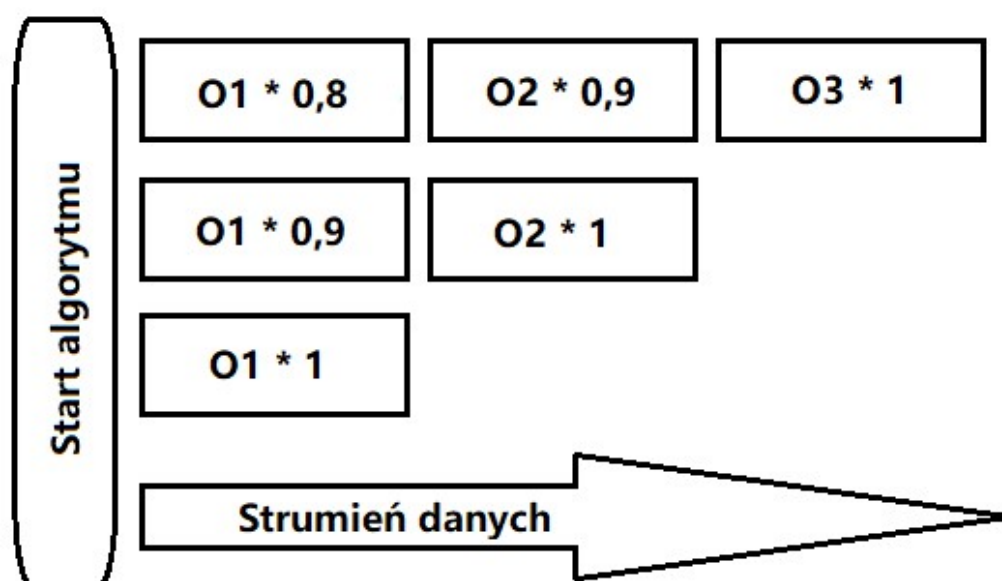
*Rysunek 1. Model Landmark Window  
Źródło: na podstawie [1]*

W tym modelu wszystkie wygenerowane dane na wyjściu są przybliżone. Innymi słowy, dane ze strumienia są skanowane od momentu rozpoczęcia analizy okna do danego punktu, w celu wydobycia z nich wiedzy. Pod uwagę brane są nie tylko najnowsze dane, ale również dane historyczne. Wsparcie dla każdego elementu jest obliczane na podstawie całego zestawu danych pomiędzy punktem orientacyjnym a bieżącym czasem. Jednak o algorytmach stosujących ten model mówi się, iż „nie są

świadome czasu”. W efekcie tracą one informacje o czasie wydobycia danych, to znaczy informacja z głębokiej przeszłości jest traktowana na równi z najnowszą. Kolejnym z minusów tego podejścia jest fakt trudności w określeniu punktu orientacyjnego. Jednocześnie wszystkie punkty w oknie są traktowane jednakowo [1].

## 2.2. Model Fading Window

Wariacją modelu Landmark jest model Fading. Model ten również bierze pod uwagę wszystkie dane od początku strumienia do aktualnego punktu w czasie. Jego wyróżnikiem jest to, iż w tym modelu nadaje się ostatnim transakcjom wyższą wartość wagi od tych transakcji, które pojawiły się w strumieniu wcześniej. W modelu Fading Window ostatnie okna są ważniejsze od poprzednich. Modele te zawierają mechanizm zapadania, który zmniejsza wagę dla starych danych [1].



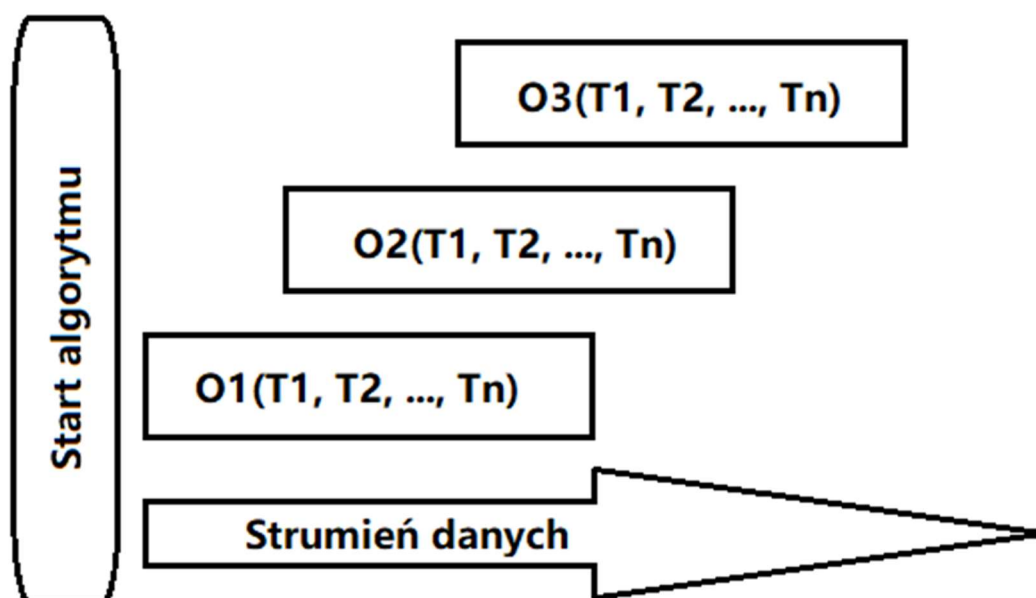
Rysunek 2. Model Fading Window  
Źródło: na podstawie [1]

Model zapadający został zaproponowany w celu przezwyciężenia ograniczeń nieświadomości czasu modelu Landmark. Podczas, gdy nowy element pojawia się w oknie, nadawana jest mu najwyższa możliwa waga, która jest redukowana w czasie zgodnie z funkcją zanikania. Szybkość zaniku jest kontrolowana przez współczynnik zanikania. W przeciwieństwie do wcześniejszego modelu, model Fading nie traktuje

jednakowo każdego elementu, starsze dane wnoszą mniej ponieważ przypisane są im mniejsze wagi. Jednak interpretacja tego modelu jest trudniejsza gdyż potencjalnie wszystkie zbiory są „aktywne”, lecz mają wagę zależną od ich wieku [3].

### 2.3. Model Sliding Window

W modelu Sliding Window okno ma z góry ustalony rozmiar od bieżącego punktu w czasie. W miarę upływu czasu, okno utrzymuje swój rozmiar i przesuwa się. Jako że okno składa się tylko z danych, które pojawiły się w transakcjach w aktualnym interwale czasowym okna, to stare dane są odrzucane. Rozmiar okna może być zdefiniowany w kategorii punktów czasowych lub ilości transakcji [3].

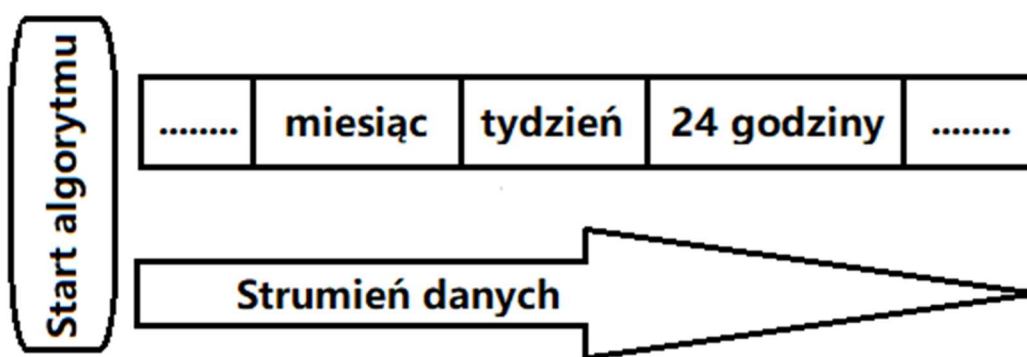


Rysunek 3. Model Sliding Window  
Źródło: na podstawie [1]

Model ten jest odpowiedni dla aplikacji, gdzie użytkownik jest zainteresowany tylko tymi danymi, które występowały ostatnio. Oczywiście w zależności od podanego rozmiaru okna można wziąć pod uwagę mniej lub więcej danych z przeszłości. Wszystkie dane w oknie przesuwnym są uważane za równie ważne. Niestety model Sliding Window ignoruje fakt, że dane mogą zmienić częstotliwość swojego występowania w transakcjach w zależności od czasu. Przez co model ten nie bierze pod uwagę ich historii występowania [1].

## 2.4. Model Time-tilted Window

Model Time-tilted Window bierze pod uwagę dane od początku strumienia do aktualnego momentu. Jednak w przypadku tego modelu okres czasu jest podzielony na konkretne przedziały. Każdy przedział czasowy odnosi się do różnego poziomu szczegółowości i posiada ograniczoną przestrzeń. Innymi słowy, przedziały czasowe w ostatnim okresie czasu są przydzielane z drobną ziarnistością, podczas gdy zakresy z dawnych czasów są przypisane z grubą ziarnistością [3].



*Rysunek 4. Model Time-tilted Window  
Źródło: na podstawie [1]*

Istnieje kilka technik okna Time-tilted, a jedną z nich jest progresywny logarytmiczny model okna Time-tilted [3]. Pozwala on na efektywną organizację przedziałów czasowych w strukturze piramidy, gdzie każdy jej poziom reprezentuje osobny zakres czasu. W rezultacie, najnowsze elementy w strumieniu są przechowywane w przedziałach o największej szczegółowości, a starsze dane zostają zapisane do przedziałów bardziej ogólnych. Intuicja podpowiada, że najnowsze dane pojawiające się w strukturze powinny być w pełni modelowane, ponieważ to one są najciekawsze, natomiast bardziej zgrubna reprezentacja powinna wystarczyć dla starszych elementów ze strumienia. Podobnie jak w modelu Fading Window model Time-tilted skupia się na najnowszych danych, bez całkowitego odrzucania starszych, jak ma to miejsce na przykład w modelu Sliding Window [3].

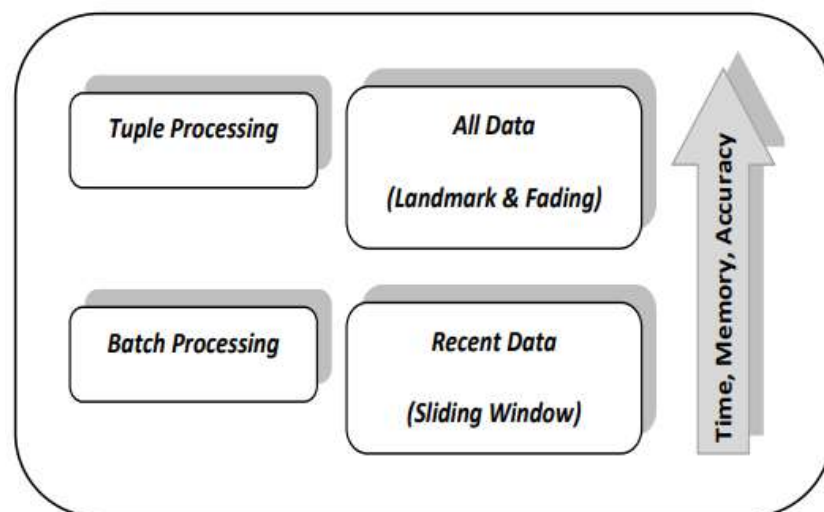


## 2.5. Kompromis między dokładnością, czasem przetwarzania i wykorzystaną pamięcią

Każdy algorytm eksplorujący strumień danych, wykorzystujący jeden z wcześniej wymienionych modeli, celuje w poprawność i dokładność wyników, minimalizowanie użycia pamięci, jednocześnie starając się wykorzystać moc procesora, ograniczając czas przetwarzania. Poprawność w tym przypadku odnosi się do eksplorowania tylko prawdziwych zbiorów częstych, a dokładność odwołuje się do analizowania dokładnej lub przybliżonej częstotliwości zbiorów. Dokładność eksploracji wymaga śledzenia wszystkich zbiorów elementów w oknie i częstotliwości ich występowania, ponieważ dowolny nieczęsty zbiór może stać się częsty w późniejszym czasie w strumieniu [1].

Jednocześnie użycie zrównoważonego progu wsparcia do kontrolowania jakości przybliżonych wyników eksploracji prowadzi do następującego dylematu: im mniejsza wartość progu, tym dokładniejsze są wyniki analizy, w konsekwencji jednak generuje to większe ilości zbiorów częstych. To z kolei prowadzi do użycia dodatkowej pamięci i mocy obliczeniowej procesora [1].

Co więcej, przetwarzanie danych odbywa się w dwóch wariantach, czyli transakcja po transakcji lub paczkach danych. Przetwarzanie pojedynczych transakcji w stosunku do całego strumienia danych, w większości przypadków jest mniej efektywne niż analizowanie paczek danych. Uogólniając, badanie paczek danych jest bardziej odpowiednie dla szybkich strumieni danych [1].



Rysunek 5. Kompromis dokładności, czasu przetwarzania i wykorzystanej pamięci  
Źródło: [1]

W dalszej części niniejszej pracy zostanie zaprezentowany algorytm estDec[2] wykorzystujący model Fading Window do wyszukiwania zbiorów częstych w strumieniach danych oraz jego implementacja w języku Java.

### 3. Algorytm estDec

Algorytm estDec wykorzystuje model Fading Window[1] do wyszukiwania zbiorów częstych w strumieniach. Metoda estDec, zgodnie z ogólnymi zasadami analizowania strumieni danych, każdą nową transakcję odczytuje i przeprowadza na niej analizę tylko raz. Również zwraca uwagę na użycie pamięci ze względu na nieskończoną naturę strumienia danych. Ponadto, już na samym początku dokumentu opisującego wyżej wymienioną metodę jej autorzy zwracają uwagę na szybkość przetwarzania transakcji. Wynik analizy powinien też być dostępny natychmiastowo, na żądanie. Ze względu na wcześniej wymienione oczekiwania, w przetwarzaniu danych dozwala się na dopuszczenie małego błędu w wynikach analizy.

Przedstawiony w tym rozdziale algorytm pochodzi z [2].

#### 3.1. Koncepcja

Strumień danych można podzielić na dwa powszechne rodzaje:

- strumień danych offline - jest to na przykład strumień danych generowany przez systemy magazynowe, gdzie każde nowo wygenerowane dane zawierają paczkę transakcji,
- strumień danych online – tego typu strumień może generować na przykład system monitorowania sieci, gdzie wszystkie generowane transakcje są dodawane do strumienia pojedynczo.

Dla strumieni danych offline możliwe jest zwiększenie wydajności eksploracji danych poprzez przeprowadzanie analizy na pakiecie danych. Z tego powodu, wyniki analizy dostępne są dopiero po zakończeniu operacji na paczce danych. W efekcie szczegółowość generowania najbardziej aktualnego wyniku zależy od liczby przetworzonych partii nowych transakcji. Natomiast eksploracja strumieni danych

online powinna wspierać elastyczny kompromis między czasem przetwarzania a dokładnością analizy bez przetwarzania danych w pakietach, aby jak najszybciej uchwycić zmianę w danych strumienia.

Generalnie wiedza zawarta w strumieniu danych zmienia się w czasie. Identyfikowanie ostatnich zmian w strumieniu online może oznaczać odkrycie wartościowej informacji. Ponadto, monitorowanie ciągłej zmienności strumienia danych pozwala zauważyć stopniowe zmienianie się wiedzy, która może zostać później odpowiednio wykorzystana. Aby to osiągnąć, należy wyeliminować wpływ przestarzałych danych na bieżący wynik analizy. Jako rozwiązanie można rozważyć model Sliding Window. Jednak rozwiązanie to bierze pod uwagę tylko te transakcje, które zostały wygenerowane w ostatnim okresie okna, co jest raczej niewyszukanym sposobem ignorowania nieaktualnych informacji. Dodatkowo wszystkie transakcje w oknie muszą być utrzymane w odpowiedni sposób, aby nie wpłynęły na wyniki analizy, gdy znajdują się poza zakresem okna.

estDec proponuje adaptacyjną metodę wyszukiwania ostatnich zbiorów częstych w strumieniu danych online. Bada ona każdą transakcję w strumieniu jedno-po-drugiej, bez generowania kandydatów. Licznik wystąpienia istotnych zbiorów jest utrzymywany w strukturze drzewa prefiksów w głównej pamięci programu. Wpływ starych transakcji na aktualny wynik eksploracji jest zmniejszony poprzez zanikanie dawnej liczby wystąpień każdego zbioru w miarę upływu czasu. Ponadto, szybkość zanikania starych danych może być elastycznie definiowana w zależności od potrzeb. Liczba znaczących zbiorów jest minimalizowana przez operacje opóźnionego wstawiania i przycinania zbioru. W rezultacie, czas przetwarzania może być elastycznie kontrolowany, w konsekwencji jednak traci się na dokładności [2]. Do zmniejszania efektu starych danych stosuje się parametr szybkości zaniku  $d$ . Gdy nowa transakcja zostaje wstawiona do odpowiedniej zmiennej w algorytmie, licznik dawnych danych zostaje obniżony przez wartość parametru  $d$ . estDec wykorzystuje tzw. regułę wyszukiwania powiązań online do oszacowywania częstotliwości występowania zbiorów [1].

### 3.2. Założenia początkowe

Przed rozpoczęciem analizy algorytmu estDec należy zdefiniować, czym jest strumień danych oraz mechanizm zanikania w kontekście tej metody.

Strumień danych może być zdefiniowany następująco:

- I. Niech  $I = \{i_1, i_2, \dots, i_n\}$  będzie zbiorem obiektów, które są jednostką informacji w aplikacji.
- II. Zbiór elementów  $e$  to taki zbiór, że  $e \in (2^I - \{\emptyset\})$ , gdzie  $2^I$  to zbiór potęgowy  $I$ . Długość  $|e|$  zbioru  $e$  oznacza liczbę elementów, z których złożony jest zbiór i oznaczony jest przez  $|e|$ -zbiór.
- III. Transakcja jest podzbiorem  $I$ . Każda transakcja ma unikalny identyfikator  $TID$ . Transakcja wygenerowana w  $k$ -tej kolejności oznaczona jest przez  $T_k$ .
- IV. Gdy nowa transakcja  $T_k$  została wygenerowana, aktualny strumień danych  $D_k$  jest złożony z wszystkich transakcji, które zostały wygenerowane do tej pory, to jest  $D_k = \{T_1, T_2, \dots, T_k\}$ , a łączna liczba transakcji w  $D_k$  jest oznaczona przez  $|D|_k$ .

Dla nowej transakcji  $T_k$ , która została wygenerowana, aktualny licznik  $C_k(e)$  zbioru elementów  $e$  jest liczbą transakcji, które zawierały dany zbiór we wszystkich  $k$  transakcjach. Podobnie aktualne wsparcie  $S_k(e)$  zbioru  $e$  jest stosunkiem jego bieżącej liczby  $C_k(e)$  do  $|D|_k$ .

Szybkość zaniku oznacza szybkość zmniejszania wagi dla ustalonej jednostki zanikania. Jednostka zaniku określa fragment informacji, która ma zaniknąć. Szybkość zanikania jest określona przez dwa parametry:  $b$  oraz  $h$ . Podstawa rozpadu, to jest  $b$ , oznacza ilość redukowanej wagi przez jednostkę rozpadu i jest większa od 1. Gdy waga aktualnej informacji jest ustawiona na 1, parametr  $h$  określa liczbę jednostek rozpadu, które sprawią, że bieżąca waga wyniesie  $b^{-1}$ . Bazując na tych dwóch parametrach, szybkość zaniku  $d$  jest zdefiniowana następująco:

$$d = b^{-\frac{1}{h}}, \text{ gdzie } b > 1, h \geq 1, b^{-1} \leq d < 1.$$

Dla danej szybkości zaniku  $d = b^{-\frac{1}{h}}$ , liczba wszystkich transakcji  $|D|_k$  w strumieniu danych  $D_k$  jest obliczana następująco:

$$|D|_k = \begin{cases} 1, & \text{jeśli } k = 1 \\ |D|_{k-1} * d + 1, & \text{jeśli } k \geq 2 \end{cases}$$

Gdy analizowana jest pierwsza transakcja, liczba transakcji  $|D|_1$  wynosi 1, ponieważ nie było wcześniej transakcji, których waga powinna zostać obniżona. Gdy podana zostaje druga transakcja, liczba transakcji  $|D|_2$  jest obliczana przez  $|D|_1 * d + 1$ , ponieważ waga pierwszej transakcji jest obniżana. Następnie dla każdej nowej transakcji generowanej w  $k$ -tej kolejności ( $k \geq 2$ ), liczba wszystkich transakcji wynosi  $|D|_k = |D|_{k-1} * d + 1$ . W konsekwencji można to wyrazić jako  $|D|_k = d^{k-1} + d^{k-2} + \dots + d + 1 = (1 - d^k)/(1 - d)$ . Ze względu na  $b^{-1} \leq d < 1$ ,  $|D|_k$  dąży do  $1/(1 - d)$  gdy  $k$  rośnie nieskończenie.

Podobnie licznik  $C_k(e)$  zbioru  $e$  w strumieniu danych można obliczyć następująco:

$$C_k(e) = C_{k-1}(e) * d + W_k(e), \text{ gdzie } W_k(e) = \begin{cases} 1, & \text{jeśli } e \in T_k \\ 0 & \end{cases}$$

### 3.3. Szacowanie licznika zbioru elementów

Nowy zbiór elementów zostaje zapisany do struktury przechowującej zbiory częste tylko w wypadku, gdy dany zbiór jest potencjalnie częsty i wszystkie jego podzbiory są obecne w strukturze. Licznik dla nowego zbioru elementów może zostać oszacowany dzięki jego podzbiорom, które są aktualnie przechowywane w drzewie zbiorów częstych. W tym celu używane są terminy zdefiniowane w poniższych definicjach.

**Definicja 1.** Dla  $n$ -zbioru  $e$  ( $n \geq 2$ ), zbiór wszystkich jego podzbiorów  $P(e)$ , zbiór wszystkich jego  $m$ -podzbiorów  $P_m(e)$  i zbiór liczników dla  $m$ -podzbiorów  $P_m^C(e)$  są zdefiniowane następująco.

- I. Zbiór podzbiorów  $P(e)$  jest złożony z wszystkich możliwych do utworzenia zbiorów z elementów zbioru  $e$ , to jest  $P(e) = \{\alpha \mid \alpha \in 2^e - \{e, \emptyset\}\}$ .
- II. Zbiór  $m$ -podzbiorów  $P_m(e)$  jest złożony z tych zbiorów z  $P(e)$ , które mają  $m$  elementów ( $m < n$ ), to jest  $P_m(e) = \{\alpha \mid \alpha \in P(e) \wedge |\alpha| = m\}$ .
- III. Zbiór liczników dla  $m$ -podzbioru  $P_m^C(e)$  jest złożony z liczników wszystkich zbiorów w  $P_m(e)$ , to jest  $P_m^C(e) = \{C(\alpha) \mid \alpha \in P_m(e)\}$ ,

gdzie  $C(e)$  oznacza wartość licznika dla zbioru elementów w strumieniu danych.

**Definicja 2.** Dla dwóch zbiorów elementów  $e_1$  i  $e_2$  ich suma  $e_1 \cup e_2$  oraz część wspólna zbiorów  $e_1 \cap e_2$  są zdefiniowane następująco:

- I. Suma zbiorów  $e_1 \cup e_2$  jest złożona z wszystkich elementów, z których złożone są  $e_1$  lub  $e_2$ .
- II. Część wspólna zbiorów  $e_1 \cap e_2$  jest złożona z tych elementów, które są obecne jednocześnie w  $e_1$  i  $e_2$ .

Dla każdego zbioru, każdy z jego podzbiorów występuje w przynajmniej tylu transakcjach co dany zbiór. Co więcej, gdy wszystkie elementy z danego zbioru występują zawsze razem, licznik danego zbioru powinien być identyczny jak w przypadku jego podzbiorów. W związku z tym, licznik zbioru zależy od tego, jak często jego elementy występują razem w transakcji. Bazując na tej obserwacji, możliwy zakres licznika zbioru można identyfikować na podstawie dwóch skrajnych rozkładów: najmniej wyłącznie rozproszonych i najbardziej wyłącznie rozproszonych. Gdy elementy zbioru są najmniej wyłącznie rozproszone występują one w jak największej liczbie transakcji. Z drugiej strony, elementy występują wyłącznie w jak największej liczbie transakcji, gdy są najbardziej wyłącznie rozproszone.

W zbiorze danych  $D$ , licznik  $n$ -zbioru  $e$  może być oszacowany przez indywidualne wartości liczników jego podzbiorów. Jego maksymalna wartość licznika  $C^{max}(e)$  może zostać odnaleziona, gdy wszystkie podzbiory są najmniej wyłącznie rozproszone. Jest to najmniejsza wartość licznika wśród podzbiorów zbioru elementów. Ponieważ zbiór jego  $(n - 1)$  – podzbiorów może dostarczyć najdokładniejszą informację o liczniku  $n$ -zbioru,  $C^{max}(e)$  można oszacować tylko na podstawie  $(n - 1)$ -podzbiorów. Dlatego, gdy  $\min(V)$  oznacza najmniejszą wartość w zbiorze wartości  $V$ , maksymalna wartość licznika  $C^{max}(e)$  zbioru  $e$  jest obliczana w następujący sposób:

$$C^{max}(e) = \min(P_{n-1}^C(e)).$$

Dla dwóch zbiorów elementów  $e_1$  i  $e_2$ , ich minimalna wartość licznika  $C^{min}(e_1 \cup e_2)$  sumy zbiorów  $e_1 \cup e_2$  może zostać oszacowana w następujący sposób:

$$C^{min}(e_1 \cup e_2) = \begin{cases} \max(0, C(e_1) + C(e_2) + C(e_1 \cap e_2)), & \text{jeśli } e_1 \cap e_2 \neq \emptyset \\ \max(0, C(e_1) + C(e_2) + |D|), & \text{jeśli } e_1 \cap e_2 = \emptyset \end{cases}$$

gdzie  $|D|$  oznacza liczbę transakcji w  $D$ , a  $\max(V)$  oznacza największą wartość w zbiorze  $V$ . Bazując na powyższym równaniu, minimalna wartość licznika  $C^{min}(e)$  zbioru elementów  $e$  może być oszacowana przez indywidualne wartości liczników jego  $(n - 1)$ -podzbioru. Innymi słowy, dla każdej osobnej pary  $(\alpha_i, \alpha_j)$   $(n - 1)$ -podzbioru, takiej że  $\alpha_i$  i  $\alpha_j$  należą do  $P_{n-1}(e)$ , licznik sumy zbiorów  $\alpha_i \cup \alpha_j$  może zostać oszacowany. Minimalną wartość licznika  $C^{min}(e)$  dla zbioru  $e$  można odnaleźć w następujący sposób:

$$C^{min}(e) = \max(\{C^{min}(\alpha_i \cup \alpha_j) \mid \forall \alpha_i, \alpha_j \in P_{n-1}(e) \wedge i \neq j\}).$$

Maksymalna wartość licznika  $C^{max}(e)$  dla zbioru  $e$  jest używana jako szacunkowa wartość licznika. W konsekwencji może istnieć błąd szacunkowy licznika, ponieważ  $C^{max}(e)$  jest największą możliwą wartością, iż zbiór pojawi się w transakcjach zbioru danych. Błąd oszacowania  $E(e)$  może zostać obliczony w następujący sposób:

$$E(e) = C^{max}(e) - C^{min}(e).$$

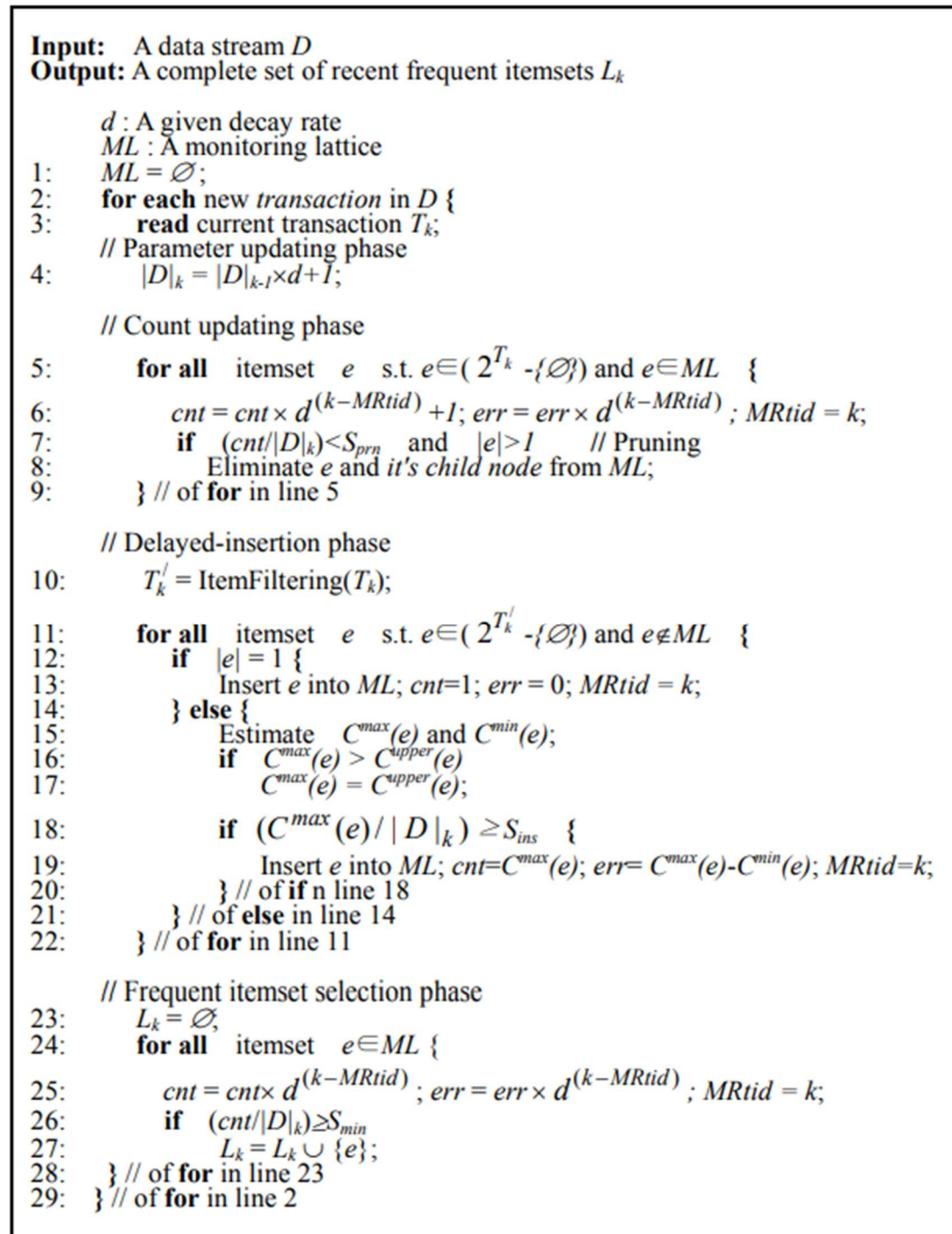
### 3.4. Metoda estDec

Nie wszystkie zbiory elementów w strumieniu danych są znaczące dla wyszukiwania zbiorów częstych. Zbiór, którego wsparcie jest mniejsze od predefiniowanego minimalnego wsparcia nie musi być nadzorowany, ponieważ nie może się stać częsty w najbliższej przyszłości. W konsekwencji, wstawianie nowego zbioru może zostać opóźnione do czasu gdy zbiór stanie się potencjalnie częsty w przyszłości. Gdy oszacowana wartość licznika dla nowego zbioru jest wystarczająco duża, zostaje on oznaczony jako „znaczący” i wstawiony do struktury przechowującej zbiory częste. Z drugiej strony, efektywny mechanizm przycinania jest następnym sposobem redukcji używanej pamięci. Pomimo, że zbiór elementów w drzewie prefiksów był na tyle znaczący, aby być nadzorowanym, może on zostać usunięty ze struktury, jeśli jego aktualne wsparcie spadnie poniżej predefiniowanego minimalnego wsparcia.

Każdy węzeł w strukturze przechowującej zbiory utrzymuje trójkę informacji  $(cnt, err, MRtid)$  dla odpowiadającego mu zbioru. Licznik zbioru  $e$  jest oznaczony

przez  $cnt$ . Maksymalny błąd licznika dla zbioru  $e$  oznaczony jest przez  $err$ . Finalnie, identyfikator najnowszej transakcji oznaczony jest przez  $MRtid$ .

Metoda estDec składa się z czterech faz: faza aktualizowania parametru (Faza I), faza aktualizowania licznika (Faza II), faza opóźnionego wstawiania (Faza III) i faza wybierania zbiorów częstych (Faza IV). Rysunek 6 przedstawia szczegółowe etapy wcześniej wymienionych faz.



Rysunek 6. Metoda estDec  
 Źródło: [2]



### 3.4.1. Faza I

Dla każdej nowo wygenerowanej transakcji  $T_k$  w strumieniu danych, liczba transakcji w aktualnym strumieniu danych  $|D|_k$  jest aktualizowana w następujący sposób:

$$|D|_k = |D|_{k-1} * d + 1.$$

### 3.4.2. Faza II

W fazie aktualizacji licznika, jego wartości - dla tych zbiorów w strukturze, które pojawiły się w nowej transakcji - są aktualizowane. Wszystkie ścieżki drzewa, które są oznaczone elementami z transakcji są przeglądane i zawarta w nich trójka informacji ( $cnt_{pre}, err_{pre}, MRtid_{pre}$ ) w każdym węźle ścieżki jest aktualizowana do aktualnej trójki ( $cnt_k, err_k, MRtid_k$ ) w następujący sposób:

$$cnt_k = cnt_{pre} * d^{(k-MRtid_{pre})} + 1,$$

$$err_k = err_{pre} * d^{(k-MRtid_{pre})},$$

$$MRtid_k = k.$$

W przypadku, gdy zaktualizowane wsparcie, to jest  $cnt_k/|D|_k$ , zbioru elementów jest mniejsze od zdefiniowanego wcześniej progu, taki zbiór przestaje być znaczący i w konsekwencji zostaje usunięty ze struktury. Jednak w przypadku  $I$ -zbioru, gdy zostanie on usunięty ze struktury, nie ma możliwości by oszacować jego licznik w przyszłości. W związku z tym, nie powinno się go usuwać. Mechanizm ten jest nazwany operacją odcinania zbioru. Próg operacji odcinania jest zdefiniowany jako próg dla odcinania  $S_{prn}$  i powinien być mniejszy od minimalnego wsparcia  $S_{min}$ .

### 3.4.3. Faza III

W następnym kroku, gdy wszystkie zbiory elementów zostały już zaktualizowane, następuje faza opóźnionego wstawiania w celu odnalezienia zbiorów, które posiadają wysokie prawdopodobieństwo stania się zbiorami częstymi w najbliższej przyszłości. Na samym początku tej fazy ma miejsce filtrowanie transakcji. Jest to operacja opcjonalna i indywidualna dla każdej implementacji metody estDec. Następnie każdy nowy zbiór elementów jest wstawiany do struktury w dwóch przypadkach. Pierwszym z nich jest, gdy w nowo wygenerowanej transakcji pojawia się jednoelementowy zbiór ( $I$ -zbiór). W tym przypadku, zbiór jest natychmiastowo wstawiany do struktury bez jakiegokolwiek procesu szacowania.

W związku z tym, wartość licznika dla  $l$ -zbioru nie jest szacowana, gdyż jest to wartość stała. Drugim przypadkiem jest  $n$ -zbiór ( $n \geq 2$ ), który nie jest jeszcze w strukturze, gdy jego oszacowane wsparcie jest wystarczająco duże, aby go monitorować. W fazie tej, wszystkie elementy, których wsparcie jest mniejsze od  $S_{ins}$ , nie są brane pod uwagę. Podczas odczytywania struktury zgodnie z pozostałymi elementami w nowej transakcji, wartość licznika nieistotnego zbioru, który składa się z znaczącego zbioru i jednego z pozostałych elementów, jest szacowana przez jego maksymalną wartość licznika  $C^{max}(e)$ . Z powodu charakterystyk struktury drzewa prefiksów, w której przechowywane są potencjalne zbiory częste, nie ma procesu generowania kandydatów. Dzieje się tak, gdyż zbiór jest systematycznie identyfikowany podczas trawersowania struktury zgodnie z pozostałymi elementami transakcji. Jeśli którykolwiek z jego  $(|e| - 1)$ -podzbiorów w  $P_{n-1}(e)$  nie jest aktualnie utrzymywany w strukturze, wartość jego licznika nie jest szacowana. Jest to spowodowane tym, iż  $C^{max}(e)$  zawsze wynosi 0 w takim przypadku. Następnie, oszacowane wsparcie dla zbioru może zostać obliczone na podstawie stosunku wartości licznika do liczby transakcji  $|D|_k$ . Jeśli obliczona wartość wsparcia jest większa lub równa predefiniowanemu progowi, zbiór jest wstawiany do struktury. Mechanizm ten jest nazwany operacją opóźnionego wstawiania, a predefiniowany próg dla wstawiania jest nazwany progiem dla opóźnionego wstawiania  $S_{ins}$  i jego wartość powinna być mniejsza od minimalnego wsparcia  $S_{min}$ .

Gdy zbiór  $e$  jest wstawiany, wszystkie jego  $(|e| - 1)$ -podzbiory powinny być znaczące. Z tego powodu, możliwe jest odnalezienie górnej granicy  $C^{upper}(e)$  wartości jego aktualnego licznika kiedy jest wstawiany w  $k$ -tej transakcji. Innymi słowy, wśród  $k$  transakcji wygenerowanych do tej pory, przynajmniej  $|e| - 1$  transakcji, które zawierały zbiór  $e$ , są wymagane, aby wstawić wszystkie jego podzbiory do struktury przed nim. W związku z tym, jego aktualna wartość licznika jest zwiększana gdy te  $|e| - 1$  transakcji zostanie ostatnio wygenerowanych. Obniżona wartość licznika zbioru  $e$  dla wstawiania jego podzbiorów przez te ostatnie  $|e| - 1$  transakcji jest reprezentowane przez  $cnt\_for\_subsets$  i obliczane w następujący sposób:

$$cnt\_for\_subsets = \{1 - d^{(|e|-1)}\} / (1 - d).$$

Dodatkowo, maksymalna możliwa obniżona wartość licznika dla zbioru  $e$  przed ostatnimi  $|e| - 1$  transakcjami oznaczona jest jako  $max\_cnt\_before\_subsets$  i oblicza się ją następująco:

$$max\_cnt\_before\_subsets = S_{ins} * |D|_{k-(|e|-1)} * d^{(|e|-1)}.$$

W konsekwencji,  $C^{upper}(e)$  może zostać obliczone następująco:

$$C^{upper}(e) = max\_cnt\_before\_subsets + cnt\_for\_subsets.$$

Jeśli  $C^{max}(e)$  jest większe od górnej granicy  $C^{upper}(e)$ ,  $C^{upper}(e)$  jest używane jako wartość licznika dla zbioru  $e$ . Odpowiednio, aktualna trójka informacji  $(cnt_k, err_k, MRtid_k)$  w zbiorze  $e$  w odpowiadającym mu węźle struktury przechowującej znaczące zbiory jest aktualizowana następująco:

$$cnt_k = \min\{C^{max}(e), C^{upper}(e)\},$$

$$err_k = E(e) = cnt_k - C^{min}(e),$$

$$MRtid_k = k.$$

Odcięty zbiór elementów może zostać wstawiony do struktury w przyszłości w operacji opóźnionego wstawiania, jeśli stanie się częsty w nowych transakcjach.  $S_{prn}$  powinno być mniejsze od  $S_{min}$ . W miarę jak różnica między progami  $S_{min}$  i  $S_{prn}$  zwiększa się, możliwość powtarzalnego odcinania i wstawiania tego samego zbioru elementów zmniejsza się. Jednocześnie w miarę zwiększania się tej samej różnicy dokładność eksploracji zbiorów częstych zwiększa się, jednocześnie zwiększając rozmiar struktury przechowującej zbiory częste.

#### 3.4.4. Faza IV

Faza wybierania zbiorów częstych jest wykonywana tylko w przypadku, gdy wynik eksploracji jest potrzebny. Tworzy on wszystkie aktualne zbiory częste z struktury danych. Podczas wykonywania tej fazy na aktualnym strumieniu danych  $D_k$ , zbiór elementów  $e$  jest częsty tylko jeśli jego aktualne wsparcie  $\{cnt * d^{(k-MRtid)}\}/|D|_k$  jest większe od predefiniowanego minimalnego wsparcia  $S_{min}$ . Również jego aktualny błąd wsparcia może zostać wyznaczony wg wzoru  $\{err * d^{(k-MRtid)}\}/|D|_k$ .

Wszystkie nieznaczące zbiory mogą zostać odcięte razem przez sprawdzanie wsparcia każdego zbioru w strukturze. Mechanizm ten, nazwany operacją

wymuszonego odcinania, może być wykonywany cyklicznie lub gdy rozmiar struktury danych osiągnie predefiniowany próg.

## 4. Symulacja działania algorytmu estDec

Wstępne wartości parametrów i danych do przeprowadzenia symulacji działania algorytmu estDec:

1.  $b = 2$ ,
2.  $h = 10000$ ,
3.  $d = b^{-\left(\frac{1}{h}\right)} = 0,99993$ ,
4.  $S_{min} = 0,5$ ,
5.  $S_{ins} = 10\% * S_{min} = 0,05$ ,
6.  $S_{prn} = S_{ins}$ ,
7.  $D = \{\{1,2\}, \{2,4\}, \{2,4\}\}$ ,
8.  $L_k = \emptyset$ ,
9.  $ML = \emptyset$ ,
10. Faza wybierania zbiorów częstych odbędzie się na końcu symulacji ze względu na to, iż nie jest wymagane przeprowadzanie jej za każdym razem.

Przebieg algorytmu:

$$k = 1$$

$$T_k = \{1,2\}$$

Faza aktualizowania parametrów

$$|D|_k = |D|_{k-1} * d + 1 = 0 * 0,99993 + 1 = 1$$

Następnie powinna nastąpić faza aktualizowania licznika, jednak żaden  $e \in \{2^{T_k} - \{\emptyset\}\}$  nie należy do  $ML$ .

Faza opóźnionego wstawiania, brak filtrowania transakcji.

Dla każdego  $e$  takiego, że  $e \in \{2^{T_k} - \{\emptyset\}\}$  i  $e \notin ML$ .

$$e = \{1\}$$

$$|e| = 1$$

$$cnt = 1, err = 0, MRtid = 1$$

$$e \rightarrow ML$$

$$ML = \{\{1\}\}$$

$$e = \{2\}$$

$$|e| = 1$$

$$cnt = 1, err = 0, MRtid = 1$$

$$e \rightarrow ML$$

$$ML = \{\{1\}, \{2\}\}$$

$$e = \{1, 2\}$$

$$|e| = 2$$

Ponieważ  $|e| \neq 1$ , należy oszacować  $C^{max}(e)$  i  $C^{min}(e)$ .

$$P_{n-1}(e) = \{\{1\}, \{2\}\}$$

$$P_{n-1}^C(e) = \{\{1\}, \{1\}\}$$

$$C^{max}(e) = \min(P_{n-1}^C(e)) = 1$$

$$C^{min}(e) = \max(\{C^{min}(\alpha_i \cup \alpha_j) \mid \forall \alpha_i, \alpha_j \in P_{n-1}(e) \wedge i \neq j\})$$

$$e_1 = \{1\}, e_2 = \{2\}$$

$$e_1 \cap e_2 = \emptyset, \text{ więc}$$

$$C^{min}(e_1 \cup e_2) = \max(0, C(e_1) + C(e_2) - |D|) = \max(0, 1 + 1 - 1) = 1$$

$$C^{upper}(e) = \{0,05 * 0 * 0,99993\} + \left\{ \frac{1 - 0,99993}{1 - 0,99993} \right\} = 0 + 1 = 1$$

$C^{max}(e)$  nie jest większe od  $C^{upper}(e)$ , więc wartość  $C^{upper}(e)$  nie będzie przypisana do  $C^{max}(e)$ .

$$\left( \frac{C^{max}(e)}{|D|_k} \right) \geq S_{ins}$$

$$\left( \frac{1}{1} \right) \geq 0,05$$

$$cnt = 1, err = 1 - 1 = 0, MRtid = 1$$

$$e \rightarrow ML$$

$$ML = \{\{1\}, \{2\}, \{1, 2\}\}$$

Koniec fazy opóźnionego wstawiania dla  $T_1$ .

$$k = 2$$

$$T_k = \{\{2\}, \{4\}\}$$

Faza aktualizowania parametrów dla  $T_2$ .

$$|D|_k = |D|_{k-1} * d + 1 = 1 * 0,99993 + 1 = 1,99993$$

Koniec fazy aktualizowania parametrów dla  $T_2$ .

Faza aktualizowania licznika dla  $T_2$ .

Dla każdego  $e$  takiego, że  $e \in \{2^{T_k} - \{\emptyset\}\}$  i  $e \in ML$ .

$$e = \{2\}$$

$$cnt = 1 * 0,99993^{(2-1)} + 1 = 1,99993$$

$$err = 0 * 0,99993^{(2-1)} = 0$$

$$MRtid = 2$$

$$\text{Czy } \left(\frac{cnt}{|D|_k}\right) < S_{prn} \text{ i } |e| = 1?$$

Nie, ponieważ  $\left(\frac{cnt}{|D|_k}\right) = 1$ . Brak odcinania dla  $e$ .

Koniec fazy aktualizowania licznika dla  $T_2$ .

Faza opóźnionego wstawiania dla  $T_2$ .

Dla każdego  $e$  takiego, że  $e \in \{2^{T_k} - \{\emptyset\}\}$  i  $e \notin ML$ .

$$e = \{4\}$$

$$|e| = 1$$

$$cnt = 1, err = 0, MRtid = 2$$

$$e \rightarrow ML$$

$$ML = \{\{1\}, \{2\}, \{4\}, \{1,2\}\}$$

$$e = \{2, 4\}$$

Ponieważ  $|e| \neq 1$  należy oszacować  $C^{max}(e)$  i  $C^{min}(e)$ .

$$P_{n-1}(e) = \{\{2\}, \{4\}\}$$

$$P_{n-1}^C(e) = \{\{1,99993\}, \{1\}\}$$

$$C^{max}(e) = 1$$

$$C^{min}(e) = \max(\{C^{min}(\alpha_i \cup \alpha_j) \mid \forall \alpha_i, \alpha_j \in P_{n-1}(e) \wedge i \neq j\})$$

$$e_1 = \{1\}, e_2 = \{2\}$$

$$e_1 \cap e_2 = \emptyset, \text{ więc}$$

$$\begin{aligned} C^{min}(e_1 \cup e_2) &= \max(0, C(e_1) + C(e_2) - |D|) = \max(0, 1,99993 + 1 - 2) \\ &= 0,99993 \end{aligned}$$

$$C^{upper}(e) = \{0,05 * 1 * 0,99993\} + \left\{ \frac{1 - 0,99993}{1 - 0,99993} \right\} = 0 + 1 = 1,04999$$

$C^{max}(e)$  nie jest większe od  $C^{upper}(e)$ , więc wartość  $C^{upper}(e)$  nie będzie przypisana do  $C^{max}(e)$ .

$$\left( \frac{C^{max}(e)}{|D|_k} \right) \geq S_{ins}$$

$$\left( \frac{1}{1,99993} \right) \geq 0,05$$

$$cnt = 1, err = 1 - 0,99993 = 0,00007, MRtid = 2$$

$$e \rightarrow ML$$

$$ML = \{\{1\}, \{2\}, \{4\}, \{1,2\}, \{2,4\}\}$$

Koniec fazy opóźnionego wstawiania dla  $T_2$ .

$$k = 3$$

$$T_k = \{2, 4\}$$

Faza aktualizowania parametrów dla  $T_3$ .

$$|D|_k = |D|_{k-1} * d + 1 = 1,99993 * 0,99993 + 1 = 2,99979$$

Koniec fazy aktualizowania parametrów dla  $T_3$ .

Faza aktualizowania licznika dla  $T_3$ .

Dla każdego  $e$  takiego, że  $e \in \{2^{T_k} - \{\emptyset\}\}$  i  $e \in ML$ .

$$e = \{2\}$$

$$cnt = 1,99993 * 0,99993^{(3-2)} + 1 = 2,99979$$

$$err = 0 * 0,99993^{(3-2)} = 0$$

$$MRtid = 3$$

$$\text{Czy } \left( \frac{cnt}{|D|_k} \right) < S_{prn} \text{ i } |e| = 1?$$

Nie, ponieważ  $\left( \frac{cnt}{|D|_k} \right) = 1$ . Brak odcinania dla  $e$ .

$$e = \{4\}$$

$$cnt = 1 * 0,99993^{(3-2)} + 1 = 1,99993$$

$$err = 0 * 0,99993^{(3-2)} = 0$$

$$MRtid = 3$$

$$\text{Czy } \left( \frac{cnt}{|D|_k} \right) < S_{prn} \text{ i } |e| = 1?$$

Nie, ponieważ  $\left( \frac{cnt}{|D|_k} \right) = 0,66669$ . Brak odcinania dla  $e$ .

$$e = \{2, 4\}$$

$$cnt = 1 * 0,99993^{(3-2)} + 1 = 1,99993$$

$$err = 0,00007 * 0,99993^{(3-2)} = 0,00007$$

$$MRtid = 3$$

$$\text{Czy } \left(\frac{cnt}{|D|_k}\right) < S_{prn} \text{ i } |e| = 1?$$

Nie, ponieważ  $\left(\frac{cnt}{|D|_k}\right) = 0,66669$ . Brak odcinania dla  $e$ .

Koniec fazy aktualizowania licznika dla  $T_3$ .

Brak fazy opóźnionego wstawiania, ponieważ każde  $e$  należy do  $ML$ .

Faza wybierania zbiorów częstych.

Dla każdego  $e \in ML$ .

$$e = \{1\}$$

$$cnt = 1 * 0,99993^{(3-1)} = 0,99986$$

$$err = 0 * 0,99993^{(3-1)} = 0$$

$$MRtid = 3$$

$$\left(\frac{cnt}{|D|_k}\right) \geq S_{min} \rightarrow (0,33330) \geq 0,5$$

Wsparcie dla  $e$  wynosi 0,33330 i jest mniejsze od  $S_{min}$ , więc  $e$  nie będzie dodane do  $L_k$ .

$$e = \{2\}$$

$$cnt = 2,99979 * 0,99993^{(3-3)} = 2,99979$$

$$err = 0 * 0,99993^{(3-3)} = 0$$

$$MRtid = 3$$

$$\left(\frac{cnt}{|D|_k}\right) \geq S_{min} \rightarrow 1 \geq 0,5$$

Ponieważ wsparcie dla  $e$  jest większe niż  $S_{min}$ , więc  $e$  zostanie dodane do  $L_k$ .

$$e = \{4\}$$

$$cnt = 1,99993 * 0,99993^{(3-3)} = 1,99993$$

$$err = 0 * 0,99993^{(3-3)} = 0$$

$$MRtid = 3$$



$$\left(\frac{cnt}{|D|_k}\right) \geq S_{min} \rightarrow 0,66669 \geq 0,5$$

Wsparcie dla  $e$  jest większe niż  $S_{min}$ , więc  $e$  zostanie dodane do  $L_k$ .

$$e = \{1, 2\}$$

$$cnt = 1 * 0,99993^{(3-1)} = 0,99986$$

$$err = 0 * 0,99993^{(3-1)} = 0$$

$$MRtid = 3$$

$$\left(\frac{cnt}{|D|_k}\right) \geq S_{min} \rightarrow (0,33330) \geq 0,5$$

Wsparcie dla  $e$  wynosi 0,33330 i jest mniejsze od  $S_{min}$ , więc  $e$  nie będzie dodane do  $L_k$ .

$$e = \{2, 4\}$$

$$cnt = 1,99993 * 0,99993^{(3-3)} = 1,99993$$

$$err = 0,00007 * 0,99993^{(3-3)} = 0,00007$$

$$MRtid = 3$$

$$\left(\frac{cnt}{|D|_k}\right) \geq S_{min} \rightarrow 0,66669 \geq 0,5$$

$$\text{Błąd} \quad \text{szacunkowy} \quad \text{wynosi} \quad \left(\frac{err}{|D|_k}\right) = 0,00002$$

Ponieważ wsparcie dla  $e$  jest większe niż  $S_{min}$ , więc  $e$  zostanie dodane do  $L_k$ .  
Koniec fazy wybierania zbiorów częstych.

Zbiory częste po przeprowadzeniu symulacji to:

- $e_1 = \{2\}$ , którego wsparcie wynosi 1, a błąd jest równy 0,
- $e_2 = \{4\}$ , którego wsparcie wynosi (0,66669), a błąd jest równy 0,
- $e_3 = \{2, 4\}$ , którego wsparcie wynosi (0,66669), a błąd jest równy 0,00002.

## 5. Implementacja algorytmu estDec

### 5.1. Wykorzystane narzędzia i technologie

#### 5.1.1. Java

Java to współbieżny, wysokopoziomowy obiektowy język programowania oparty na klasach. Opracowany przez Jamesa Goslinga w firmie Sun z myślą o możliwości uruchamiania programów na różnych architekturach procesorów („napisz raz, uruchom wszędzie”). Java kompiluje kod źródłowy do kodu pośredniego, który następnie jest interpretowany przez maszynę wirtualną Javy do kodu dostosowanego dla odpowiedniego systemu. Pierwsza wersja Javy została wydana w 1996r [4]. Na przestrzeni lat Java znacząco ewoluowała i aktualnie jest jednym z najpopularniejszych języków programowania [WWW, 6]. Twórcy Javy zredagowali białą księgę, w której zawarli swoje idee i dokonania. Następnie wydane zostało streszczenie, które zostało „(...) zorganizowane według następujących 11 słów kluczowych:

1. *prosty,*
2. *obiektowy,*
3. *sieciowy,*
4. *niezawodny,*
5. *bezpieczny,*
6. *niezależny od architektury,*
7. *przenośny,*
8. *interpretowany,*
9. *wysokowydajny,*
10. *wielowątkowy,*
11. *dynamiczny.” [4]*

#### 5.1.2. JavaFX

JavaFX to rekomendowana biblioteka do tworzenia graficznego interfejsu użytkownika w Javie. Biblioteka ta pozwala tworzyć wydajne interfejsy z zachowaniem czytelnego kodu, które wyglądają współcześnie. Najważniejsze udoskonalenia względem swojego poprzednika, którym był Swing, to:

1. Opisywanie wyglądu aplikacji za pomocą znaczników XML. Zapewnia to łatwą edycję interfejsu oraz możliwość rozdzielania zadań.
2. Możliwość edytowania kontrolek za pomocą stylów CSS. Dzięki temu wystarczy stworzyć jeden plik przechowujący, który przechowuje cechy elementów, zamiast nadawania wyglądów w kodzie programu.
3. Wsparcie dla multimediiów. JavaFX pozwala odtwarzać pliki MP3 oraz filmy w niektórych formatach.
4. W razie konieczności wykorzystania przez dewelopera elementu Swinga, JavaFX umożliwia posługiwanie się nim w aplikacji [5].

### **5.1.3. JavaFX Scene Builder**

JavaFX Scene Builder to narzędzie umożliwiające graficzne tworzenie interfejsu użytkownika dla JavaFX. Dodawanie elementów do okna tworzonej aplikacji jest realizowane za pomocą mechanizmu drag'n'drop. Dla każdego elementu możliwe jest edytowanie jego parametrów. Po dokonaniu zapisu program wygeneruje plik z kodem XML, który można następnie formatować w edytorze. Jeśli użytkownik wprowadzi zmiany w kodzie XML w aktualnie edytowanym pliku, JavaFX Scene Builder automatycznie odświeży zmiany w swoim interfejsie [WWW, 7].

### **5.1.4. Apache NetBeans IDE**

NetBeans IDE (ang. *Integrated Development Environment*) to zintegrowane środowisko programistyczne dla języka Java wytwarzane przez Apache Software Foundation. Oparte jest na licencji Apache 2.0. Wspiera rozwijanie wszystkich typów aplikacji Java. Wśród wszystkich oferowanych funkcji NetBeans, są między innymi: wsparcie dla projektów Maven, refaktoryzacja kodu oraz kontrola wersji. Poza wsparciem dla języka Java, NetBeans oferuje również wsparcie dla C, C++, PHP oraz HTML [WWW, 8].

### 5.1.5. Spring Boot

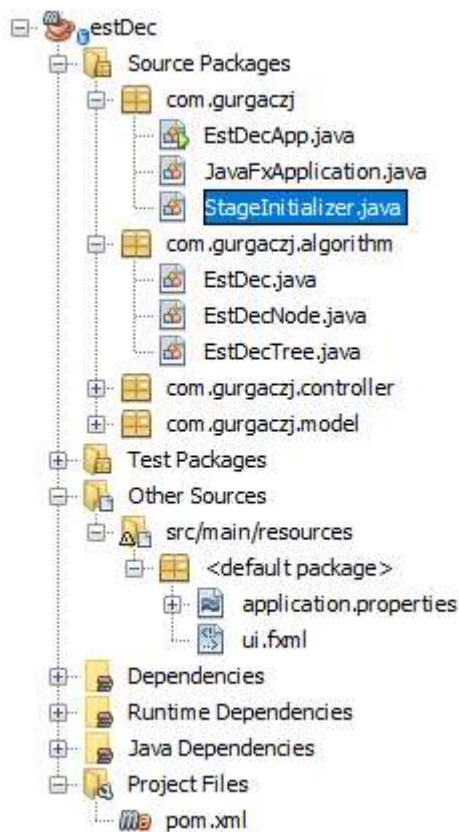
Spring to framework open-source stworzony na podstawie kodu Roda Johnsona opisanego w jego książce „*Expert One-on-One: J2EE Design and Development*”. Spring jest odpowiedzialny za wiele funkcji, jednak ogólnikowo jest to lekki kontener do wstrzykiwania zależności i programowania aspektowego. Składa się on z wielu modułów, jednak to deweloperowi pozostawia się wolność wyboru. Moduły te zapewniają niezbędne elementy do rozwoju aplikacji, najczęściej po stronie serwera [WWW, 12].

## 5.2. Struktura projektu

Implementacja algorytmu estDec została utworzona jako projekt Maven. Dzięki temu nie ma potrzeby pobierania zależności potrzebnych w projekcie ręcznie, gdyż Maven zajmuje się tym automatycznie. Jak każdy z projektów Maven, ten także zawiera plik *pom.xml*, który uwzględnia informacje o wszystkich potrzebnych zależnościach. Posiada także informacje potrzebne do kompilowania wykonywalnego pliku *.jar*. Zależności użyte w projekcie to:

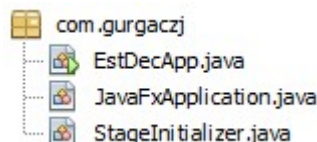
- javafx-fxml,
- javafx-base,
- javafx-graphics,
- javafx-controls,
- guava,
- spring-boot-starter.

Aplikacja składa się z czterech pakietów, gdzie każdy pakiet odpowiada innej funkcjonalności.



Rysunek 7. Struktura projektu

- **Pakiet `com.gurgaczj`**

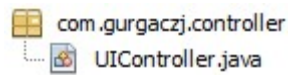


Rysunek 8. Zawartość pakietu `com.gurgaczj`

Kod zawarty w plikach `.java` w pakiecie `com.gurgaczj` odpowiada za uruchamianie aplikacji, utworzenie okna z interfejsem użytkownika i uruchomienie kontekstu Springa. Głównym plikiem, zawierającym metodę `main`, jest `EstDecApp.java`. Metoda `main` wywołuje metodę `start` w klasie `JavaFxApplication`, która dziedziczy po klasie `Application`. W metodzie `start`, za pomocą kontekstu Springa, publikowane jest zdarzenie `StageReadyEvent` z parametrem, którym jest okno aplikacji. Następnie zdarzenie jest przechwytywane przez klasę `StageInitializer`, implementującą interfejs `ApplicationListener`, który z kolei nasłuchuje zdarzeń `StageReadyEvent`. Wreszcie, metoda `onApplicationEvent` inicjalizuje interfejs użytkownika w oknie podanym jako parametr do `StageReadyEvent`. Wykorzystanie frameworka Spring w tym momencie jest zakończone, ponieważ został on użyty

w aplikacji w głównej mierze do bezproblemowego budowania aplikacji do uruchamialnego pliku *.jar*.

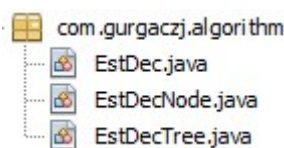
- **Pakiet *com.gurgaczj.controller***



Rysunek 9. Zawartość pakietu *com.gurgaczj.algorithm*

W pakiecie *com.gurgaczj.controller* zawarta jest klasa *UIController*, która jest odpowiedzialna za zarządzanie kontrolkami interfejsu użytkownika. Zdefiniowane są w niej zdarzenia, które odpowiadają odpowiednim przyciskom oraz sposób ich zachowania. *UIController* również odpowiada za działanie tabeli, do której dodawane są zbiory częste oraz inicjalizację algorytmu *estDec*.

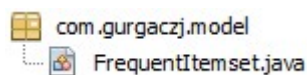
- **Pakiet *com.gurgaczj.algorithm***



Rysunek 10. Zawartość pakietu *com.gurgaczj.algorithm*

Pakiet *com.gurgaczj.algorithm* zawiera klasy odpowiedzialne za działanie algorytmu *estDec*. Pierwsza klasa *EstDec* jest klasą inicjalizującą algorytm. Zawiera ona metody, które odpowiadają za zdefiniowanie szybkości zaniku, przetwarzanie transakcji oraz wydobywanie zbiorów częstych. *EstDecNode* jest odzwierciedleniem węzła, w którym monitorowane są potencjalne zbiory częste wraz z ich wartościami licznika, błędu oraz numerem transakcji. Ostatnia klasa *EstDecTree* odpowiada za wykonanie każdej fazy algorytmu *estDec*.

- **Pakiet *com.gurgaczj.model***



Rysunek 11. Zawartość pakietu *com.gurgaczj.model*

Ostatni pakiet *com.gurgaczj.model* zawiera jedną klasę *FrequentItemset*, która opisuje zbiór częsty, jego wsparcie a także błąd.

## 5.3. estDec – implementacja w Javie

### 5.3.1. Charakterystyka systemu

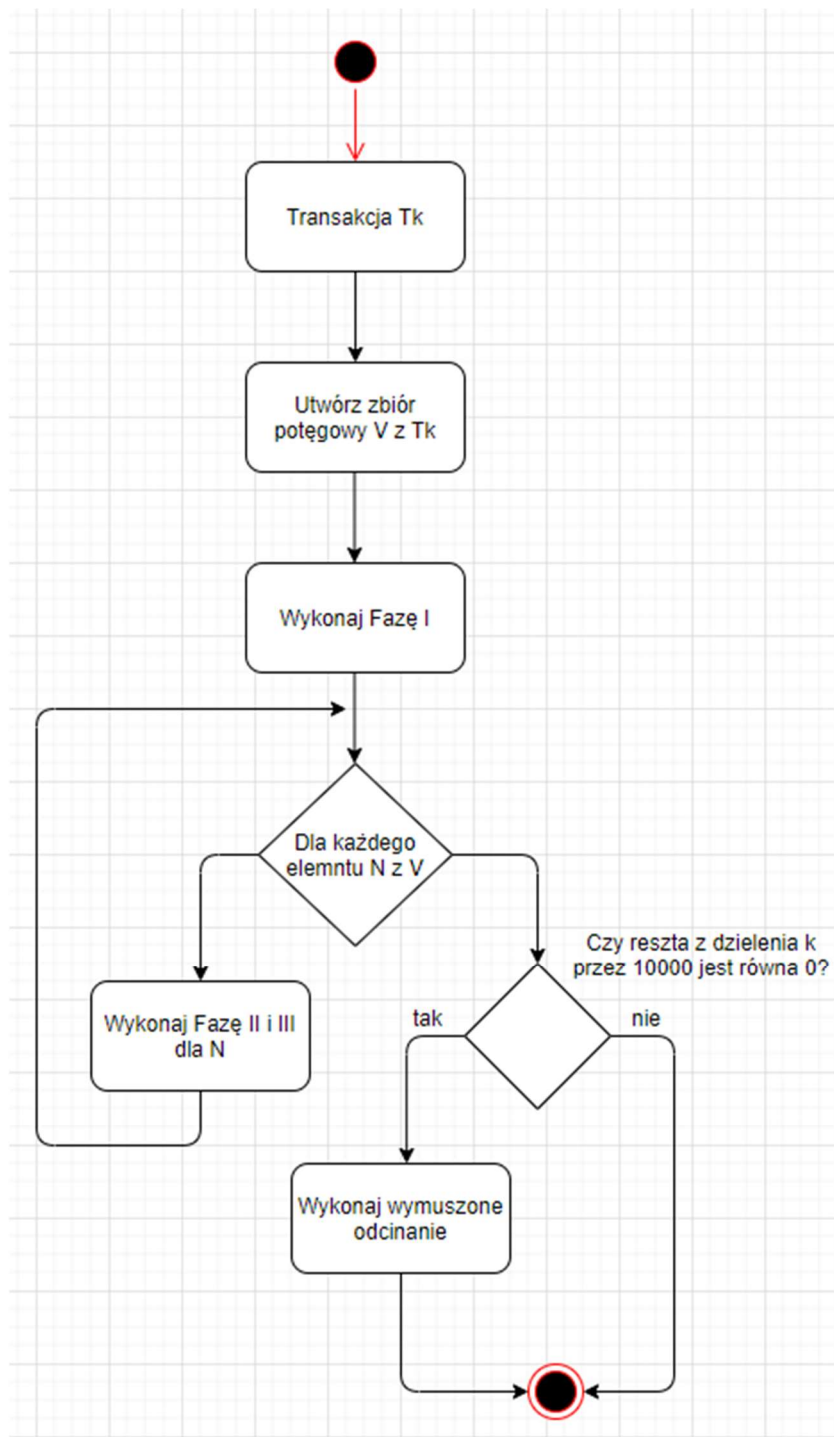
Algorytm estDec został w całości zaimplementowany w języku Java przy pomocy biblioteki Google Guava wykorzystanej na licencji Apache 2.0 oraz narzędzi opisanych w rozdziale pod tytułem: *5.1. Wykorzystane narzędzia i technologie*.

Aplikacja symuluje strumień danych poprzez wczytanie do pamięci pliku .csv, a następnie dane z pamięci są podane do algorytmu i przetwarzane. Po zakończeniu analizy zbiory częste zostają wylistowane do tabeli w interfejsie użytkownika. Jedna linia z pliku .csv oznacza jedną transakcję.

### Implementacja

Cała funkcjonalność algorytmu estDec jest zaimplementowana w pakiecie *com.gurgaczj.algorithm*. Przed rozpoczęciem analizowania danych z wybranego pliku .csv należy podać delimiter, czyli znak, którym dzielone są dane w pojedynczej linii w pliku. Następnie wszystkie dane z pliku są wczytywane do pamięci.

Każda transakcja jest podawana do algorytmu i analizowana. Pierwszym krokiem w implementacji jest utworzenie wszystkich możliwych podzbiorów (zbiór potęgowy) dla danych w danej transakcji bez zbioru pustego. Do tworzenia zbioru potęgowego dla transakcji używana jest biblioteka *Guava*. Następnie zaczyna się Faza I algorytmu, czyli aktualizowanie parametru, a po tej fazie dla każdego zbioru w zbiorze potęgowym transakcji następuje Faza II i III. Gdy analiza wszystkich podzbiorów transakcji zostanie zakończona ma miejsce mechanizm wymuszonego odcinania. Jest on jednak uruchamiany co dziesięć tysięcy transakcji.

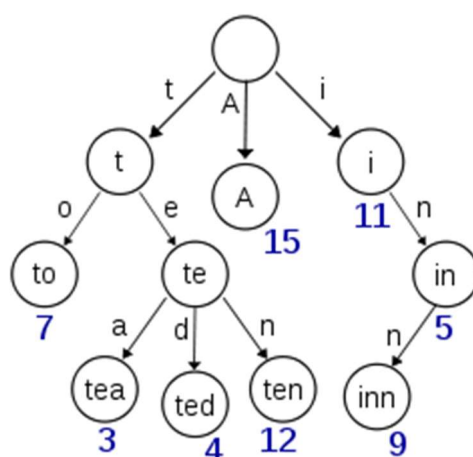


Rysunek 12. Diagram aktywności przetwarzania transakcji

W ramach przetwarzania transakcji wywoływana jest faza aktualizowania licznika i opóźnionego wstawiania, których implementacja zostanie przedstawiona poniżej. Jednak przed ich przedstawieniem, należy wyjaśnić, w jaki sposób przechowywane są potencjalnie częste zbiory.



Zbiory potencjalnie częste są przechowywane w strukturze przypominającej drzewo *trie*, gdzie korzeniem jest pusty element, a węzły od korzenia zaczynają przyjmować wartości. Wartości kluczy to elementy transakcji, które tworzą strukturę drzewa. Wszystkie ścieżki w strukturze drzewa formują elementy potencjalnie częstego zbioru oraz kolejność, w jakiej zostały dodane. Oprócz kluczy struktura przechowuje również dane o wartości licznika, błędzie szacowania licznika oraz identyfikator transakcji danego potencjalnie częstego zbioru.

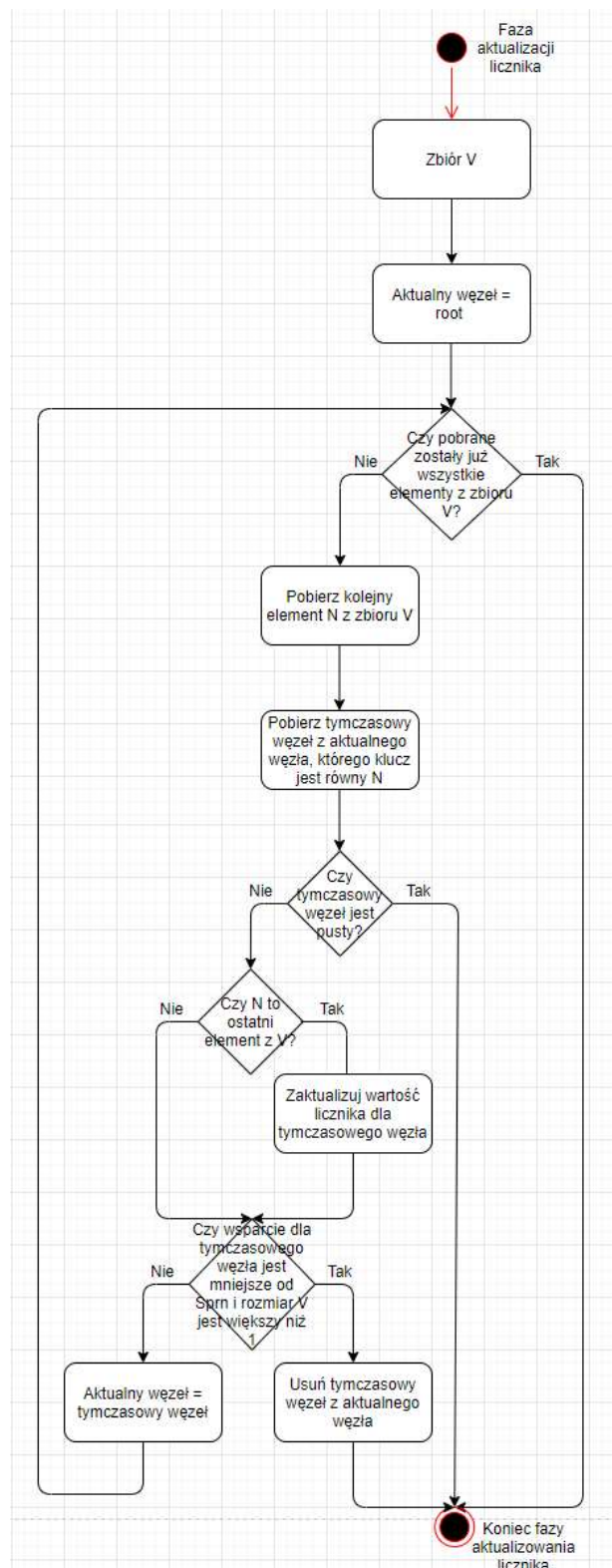


Rysunek 13. Drzewo trie

Źródło: <https://en.wikipedia.org/wiki/Trie>

Faza II metody *estDec*, czyli aktualizacja licznika, została zaimplementowana w klasie *EstDecTree*. Do funkcji aktualizującej wartość licznika jako parametr podawany jest zbiór. Następnie dla każdego elementu zbioru sprawdzane jest, czy jest on obecny w strukturze drzewa, rozpoczynając od dzieci korzenia, idąc w głąb wraz z następnymi elementami zbioru. W przypadku, gdy dany element zbioru nie istnieje jako dziecko w ścieżce, w której powinien wystąpić, aktualizacja licznika jest przerywana. Dzieje się tak, ponieważ nie ma ułożonych kluczy w drzewie w taki sposób, jak kolejność elementów w zbiorze. Gdy jednak uda się odnaleźć odpowiadającą ścieżkę kluczy w drzewie - która odpowiada kolejności elementów w zbiorze - to wartość licznika jest aktualizowana tylko dla węzła, który odpowiada ostatniemu elementowi ze zbioru. Jest to spowodowane faktem, iż wszystkie węzły drzewa prowadzące do danego klucza formują potencjalnie częsty zbiór. W efekcie aktualizacja licznika w innym węźle spowodowałaby zmianę wartości dla innego potencjalnego zbioru, który nie odpowiadałby zbiorowi podanemu w parametrze.

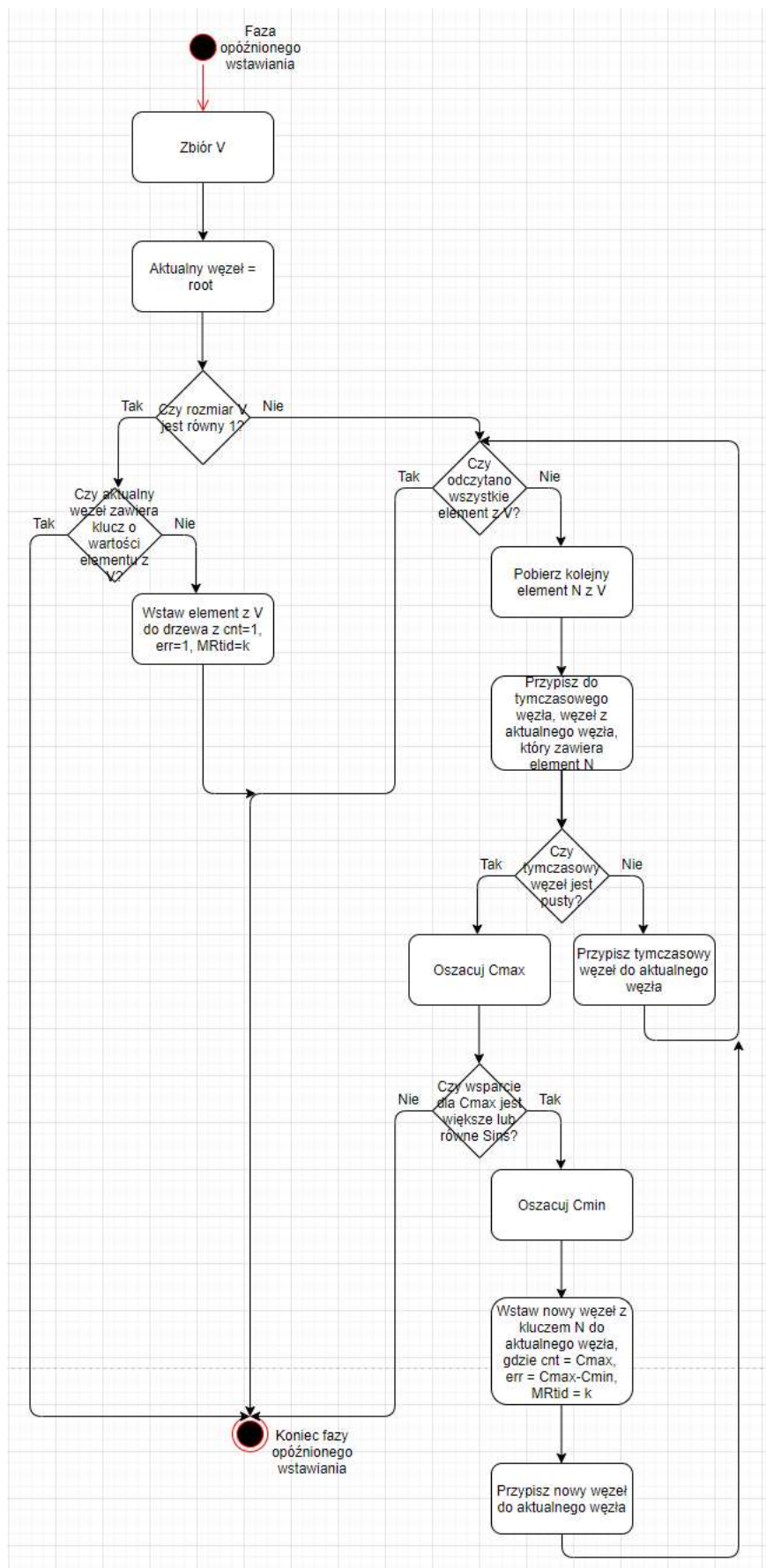
Jednocześnie, podczas przechodzenia przez węzły, obliczana jest jego wartość wsparcia. Jeśli wartość wsparcia jest mniejsza od  $S_{prn}$  i ilość elementów transakcji jest większa od 1 to dany węzeł jest usuwany z drzewa.



Rysunek 14. Diagram aktywności aktualizowania licznika

Faza opóźnionego wstawiania jest kolejnym krokiem w implementacji algorytmu `estDec`. Funkcja wstawiania elementu zbioru do drzewa jako parametr przyjmuje zbiór. Na początku implementacji opóźnionego wstawiania sprawdzany jest warunek, czy zbiór jest jednoelementowy, a następnie, czy element istnieje jako klucz w korzeniu drzewa. Jeśli dany element nie istnieje w korzeniu, zgodnie z metodą `estDec` jest on wstawiany do korzenia z wartością licznika 1 i funkcja jest zakończona. W drugim przypadku, czyli ilość elementów zbioru jest większa od 1, rozpoczyna się proces iteracji po elementach zbioru i sprawdzanie, czy elementy zbioru istnieją w ścieżce drzewa rozpoczynając od korzenia.

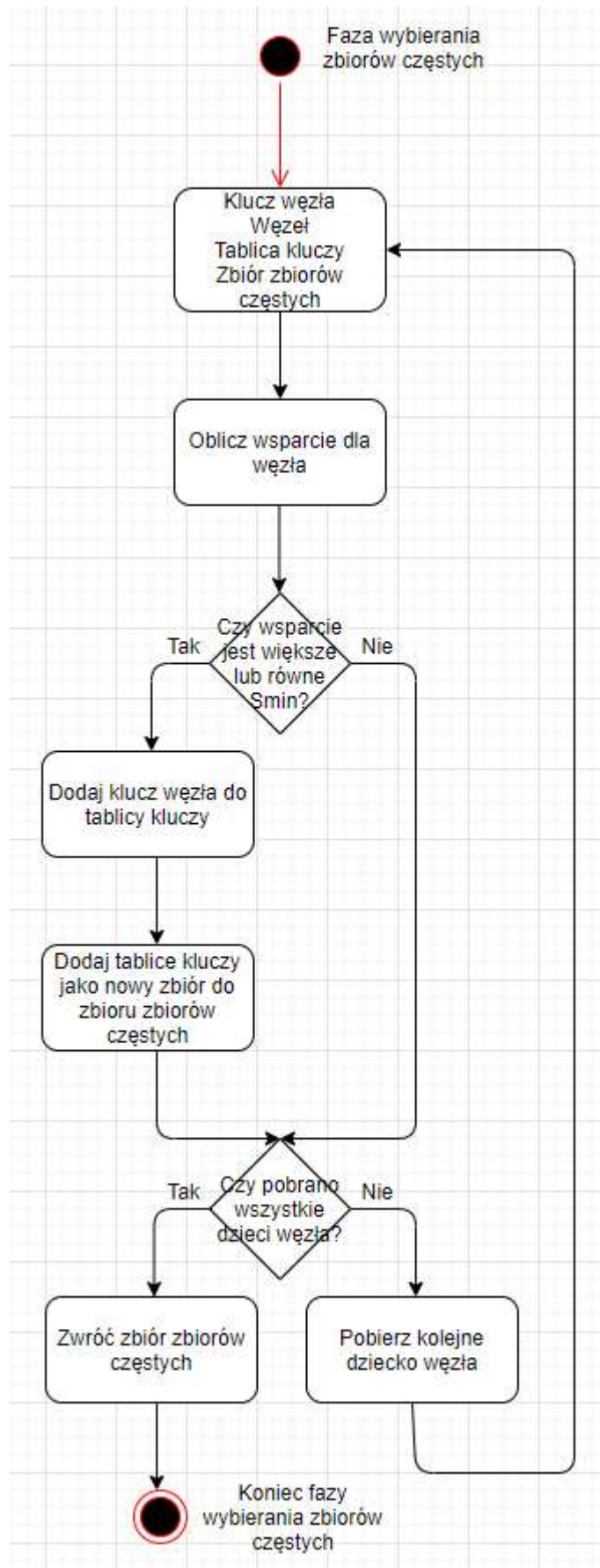
Schemat jest podobny do opisanego wcześniej w przypadku implementacji fazy aktualizowania licznika. Jednak tutaj, gdy element nie istnieje jako dziecko danego węzła, następuje próba wstawiania go do drzewa. W pierwszym kroku próby wstawienia elementu do drzewa dla zbioru, którego ilość elementów jest większa niż 1 szacowana jest wartość  $C^{max}(e)$ . Po procesie szacowania  $C^{max}(e)$ , obliczone zostaje wsparcie dla oszacowanej wartości licznika. Jeśli wartość wsparcia jest większa lub równa  $S_{ins}$  rozpoczyna się proces szacowania wartości  $C^{min}(e)$ . Po obliczeniu  $C^{min}(e)$  dany element zbioru jest wstawiany jako nowy klucz do drzewa w odpowiadającym mu węźle.



Rysunek 15. Diagram aktywności opóźnionego wstawiania

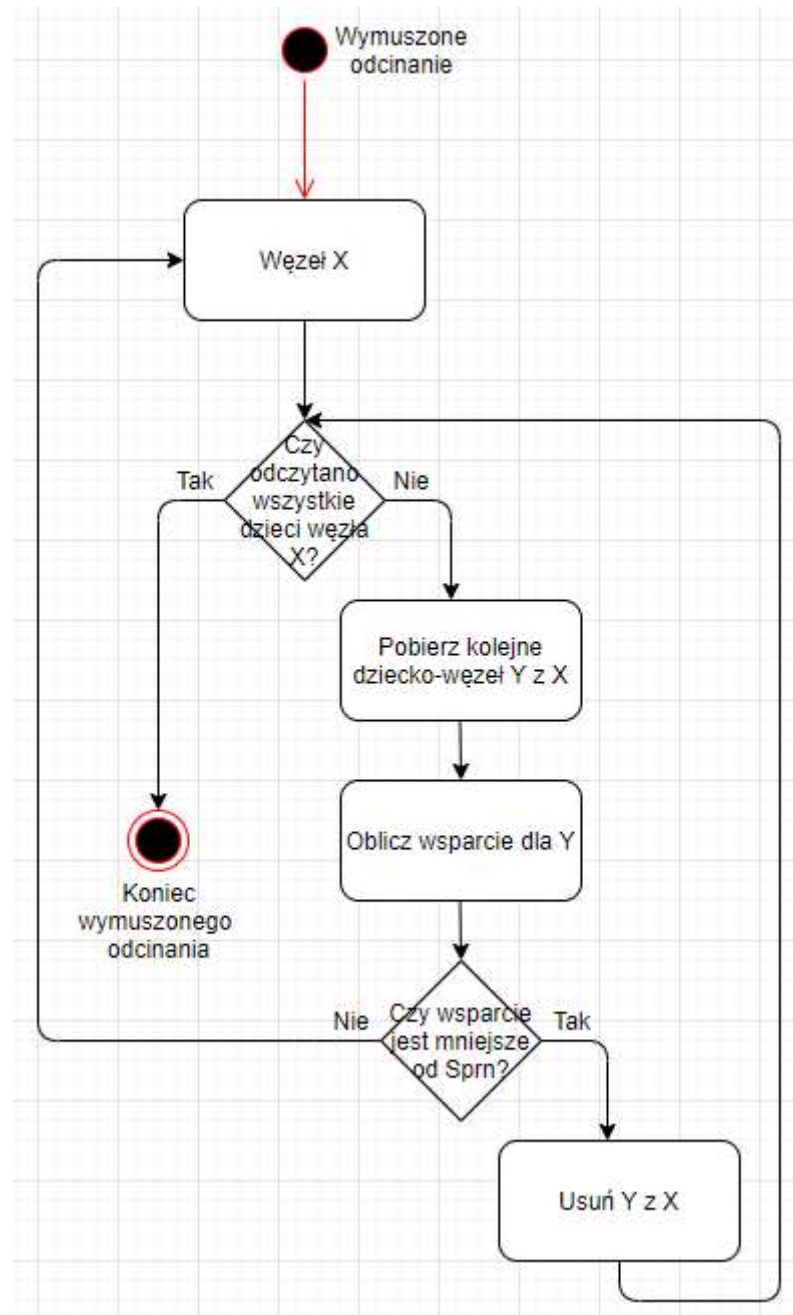
Kolejną częścią implementacji algorytmu estDec jest faza wybierania zbiorów częstych. Głównym elementem implementacji tej fazy jest rekurencyjne przechodzenie po węzłach drzewa. Funkcja realizująca to zadanie jako parametr przyjmuje klucz węzła, węzeł, tablicę wcześniejszych kluczy oraz zbiór zbiorów częstych. W pierwszym kroku aktualizowana jest wartość licznika, błędu i numeru transakcji dla węzła zgodnie z specyfikacją metody estDec. Następnie obliczane jest wsparcie dla węzła i porównywane jest z  $S_{min}$ . Jeśli wsparcie jest większe od  $S_{min}$ , klucz węzła zostaje dodany do tablicy kluczy, a do zbioru zbiorów częstych zostaje dodany zbiór, którego elementy to tablica kluczy. W następnej kolejności, dla wszystkich dzieci węzła zostaje wywołana rekurencyjnie funkcja wybierania zbiorów częstych. Dzięki rekurencji możliwe jest przeglądnięcie wszystkich węzłów drzewa w poszukiwaniu zbiorów częstych.

Funkcja rekurencyjnego wyszukiwania zbiorów częstych jest wywoływana przez funkcję, która rozpoczyna proces rekurencji podając w pierwszym wywołaniu kolejno dane każdego węzła-dziecka korzenia.



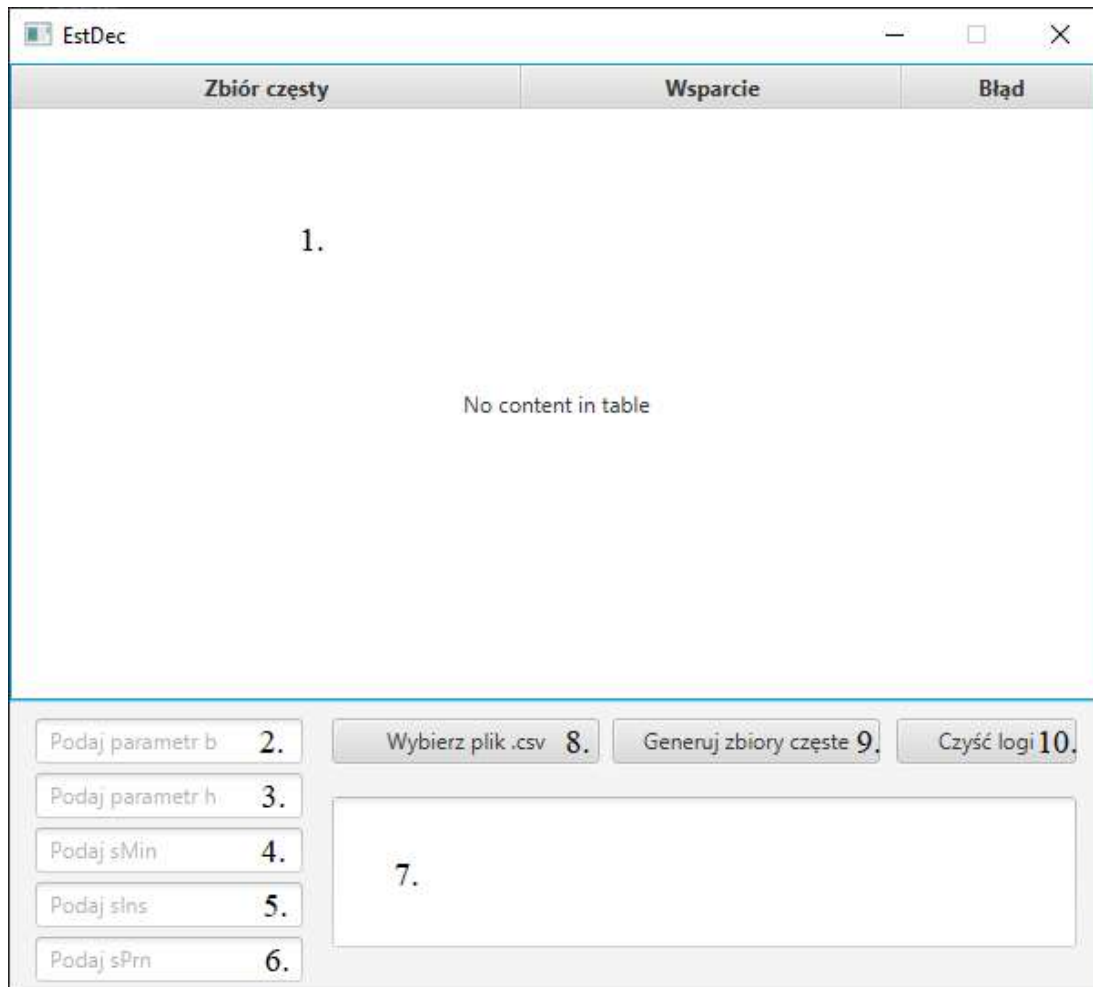
Rysunek 16. Diagram aktywności wybierania zbiorów częstych

Ostatnim elementem implementacji estDec jest wymuszone odcinanie. W tym przypadku jak i w implementacji wybierania zbiorów częstych, przeglądanie wszystkich węzłów drzewa zostało zrealizowane przy pomocy rekurencji. Podczas przeglądania, obliczane jest wsparcie dla każdego dziecka-węzła aktualnego węzła podanego w parametrze funkcji i porównywane z  $S_{prn}$ . Jeśli wartość wsparcia jest mniejsza niż  $S_{prn}$  węzeł jest usuwany z drzewa wraz z wszystkimi jego dziećmi.



Rysunek 17. Diagram aktywności wymuszonego odcinania

## 5.4. Interfejs użytkownika



Rysunek 18. Interfejs użytkownika

Interfejs użytkownika to pojedyncze okno, na które składa się dziesięć elementów:

1. Tabela zbiorów częstych. Tabela ta, zawiera trzy kolumny. W pierwszej kolumnie wyświetlane są zbiory częste. W kolejnej wsparcie dla danego zbioru. Błąd szacowania wyświetlany jest w ostatniej kolumnie.
2. Pole tekstowe, w którym należy podać wartość parametru  $b$ .
3. Pole tekstowe, w którym należy podać wartość parametru  $h$ .
4. Pole tekstowe, w którym należy podać wartość parametru  $S_{min}$ .
5. Pole tekstowe, w którym należy podać wartość parametru  $S_{ins}$ .



6. Pole tekstowe, w którym należy podać wartość parametru  $S_{prn}$ .
7. Obszar, w którym wyświetlane są logi aplikacji.
8. Przycisk do wyboru pliku `.csv`, z którego mają zostać zaimportowane dane.
9. Przycisk rozpoczyna fazę wybierania zbiorów częstych. Następnie zbiory te są wyświetlane w tabeli.
10. Przycisk do czyszczenia pola z logami.

## 5.5. Testy aplikacji

Pierwszy test aplikacji jest odzwierciedleniem przeprowadzonej symulacji algorytmu `estDec` z rozdziału 4. Na potrzebę tego testu przygotowany został plik `.csv` z danymi odpowiadającymi transakcjom z symulacji.

1	1;2
2	2;4
3	2;4

*Rysunek 19. Dane przygotowane na potrzeby pierwszego testu*



powtarzającej się transakcji bez zbioru pustego z takimi samymi wartościami wsparcia. Dzieje się tak, ponieważ wszystkie podzbiory powtarzającej się transakcji zawsze występują razem

Zbiór częsty	Wsparcie	Błąd
[częstych, w, danych, implementacja, algorytmu]	0.7597469266479578431372488012...	0
[odnajdywanie, zbiorów, w, danych, implementacja]	0.7597469266479578431372488012...	0
[zbiorów, częstych]	0.7597469266479578431372488012...	0
[zbiorów, w, danych, estDec]	0.7597469266479578431372488012...	0
[odnajdywanie, w, strumieniach, danych]	0.7597469266479578431372488012...	0
[częstych, w, danych, implementacja, algorytmu, e...]	0.7597469266479578431372488012...	0
[zbiorów, danych, implementacja]	0.7597469266479578431372488012...	0
[odnajdywanie, zbiorów, częstych, danych, estDec]	0.7597469266479578431372488012...	0
[odnajdywanie, zbiorów, strumieniach, algorytmu]	0.7597469266479578431372488012...	0
[zbiorów, strumieniach, algorytmu]	0.7597469266479578431372488012...	0
[odnajdywanie, zbiorów, strumieniach, danych, im...]	0.7597469266479578431372488012...	0
[odnajdywanie, w, danych, algorytmu]	0.7597469266479578431372488012...	0
[w, strumieniach]	0.7597469266479578431372488012...	0

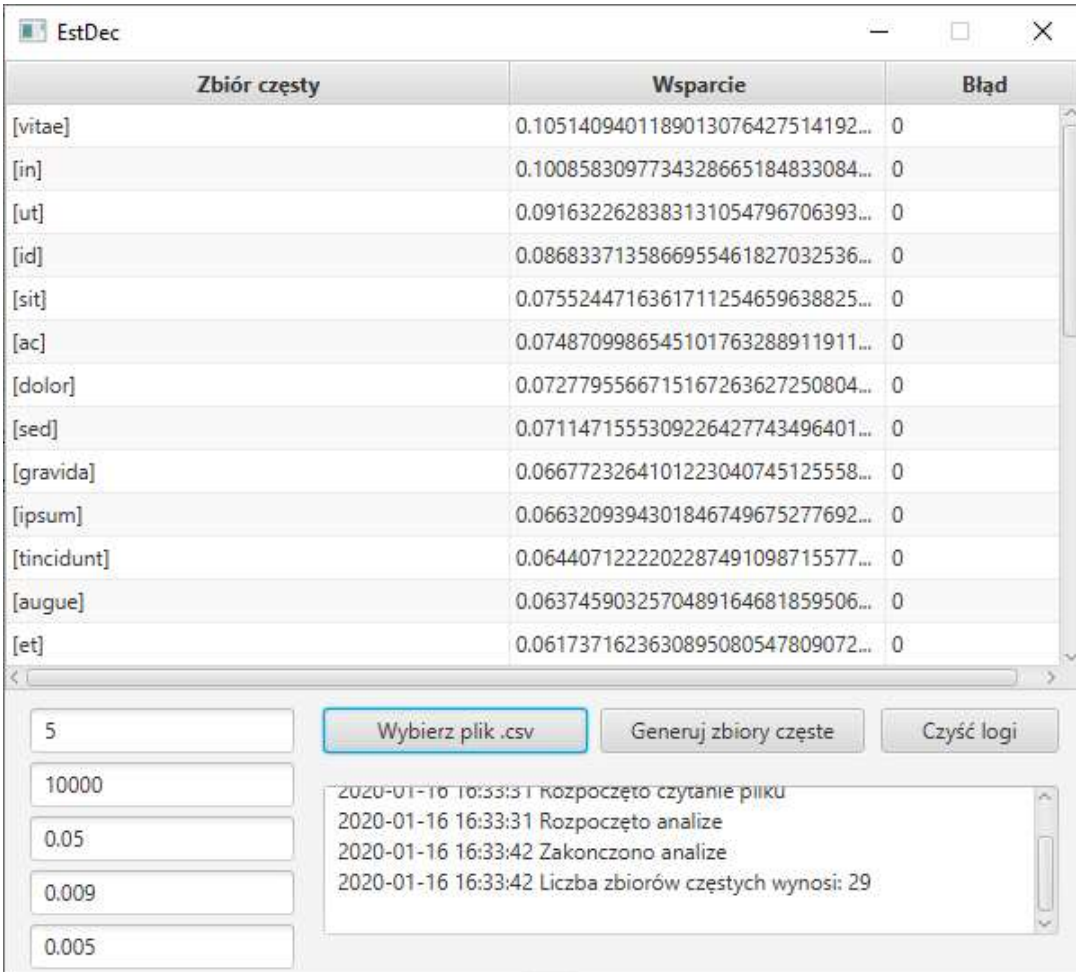
2020-01-16 00:19:24 Rozpoczęto czytanie pliku  
 2020-01-16 00:19:24 Rozpoczęto analize  
 2020-01-16 00:19:33 Zakonczono analize  
 2020-01-16 00:19:33 Liczba zbiorów częstych wynosi: 511

Rysunek 21. Wyniki drugiego testu

Zgodnie z przewidywaniami, największe wsparcie mają wszystkie możliwe do utworzenia podzbiory powtarzającej się transakcji. Dla drugiego testu zostały zmienione wartości dla parametru  $b$  i  $h$  przez co ostatnie częste zbiory mają wyższe wsparcie i jednocześnie nieczęste zbiory zanikają szybciej.

Ostatnim testem będzie poddanie analizie pliku z całkowicie losowymi danymi. Generowanie danych będzie podobne do tego z drugiego testu, jednak w tym przypadku żadna transakcja nie będzie powtarzana z zamysłem, a każda transakcja będzie losowa. Ze względu na analizowanie losowych danych szybkość zanikania zostanie ograniczona nadając wysoką wartość parametrowi  $h$ . Również parametrowi

minimalnego wsparcia zostanie przydzielona niewielka wartość, ponieważ losowe dane najprawdopodobniej nie będą posiadały wielu powtarzających się wzorów.



The screenshot shows the EstDec application window. It contains a table with three columns: 'Zbiór częsty' (Frequent Set), 'Wsparcie' (Support), and 'Błąd' (Error). The table lists 13 frequent itemsets with their corresponding support values and error values (all 0). Below the table, there are input fields for parameters: 5, 10000, 0.05, 0.009, and 0.005. There are also buttons for 'Wybierz plik .csv', 'Generuj zbiory częste', and 'Czyść logi'. A log window at the bottom right shows the execution progress.

Zbiór częsty	Wsparcie	Błąd
[vitae]	0.1051409401189013076427514192...	0
[in]	0.1008583097734328665184833084...	0
[ut]	0.0916322628383131054796706393...	0
[id]	0.0868337135866955461827032536...	0
[sit]	0.0755244716361711254659638825...	0
[ac]	0.0748709986545101763288911911...	0
[dolor]	0.0727795566715167263627250804...	0
[sed]	0.0711471555309226427743496401...	0
[gravida]	0.0667723264101223040745125558...	0
[ipsum]	0.0663209394301846749675277692...	0
[tincidunt]	0.0644071222202287491098715577...	0
[augue]	0.0637459032570489164681859506...	0
[et]	0.0617371623630895080547809072...	0

Log window content:

```

2020-01-16 16:33:31 Rozpoczęto czytanie pliku
2020-01-16 16:33:31 Rozpoczęto analize
2020-01-16 16:33:42 Zakonczono analize
2020-01-16 16:33:42 Liczba zbiorów częstych wynosi: 29

```

Rysunek 22. Wyniki trzeciego testu

W trzecim teście, wsparcie dla zbiorów częstych jest niskie ze względu na niewiele powtarzających się podzbiorów w transakcjach. W efekcie, największe wsparcie mają pojedyncze elementy.

## 6. Podsumowanie i wnioski końcowe

Głównym celem niniejszej pracy było przedstawienie modeli eksploracji danych w strumieniach danych, w szczególności zaś koncepcji do wyszukiwania zbiorów częstych zawartych w strumieniach.

Obecnie istnieją różne modele, które biorą pod uwagę wszystkie dane z strumienia lub tylko ostatnie a także adaptują mechanizm zanikania starych danych na rzecz nowych.

Zastosowanie każdego z modeli zależy od wymagań stawianych aplikacji lub algorytmowi. Model Landmark Window powinien zostać użyty jeśli wymagane jest poznanie wszystkich zbiorów częstych w strumieniu danych. Model Sliding Window najlepiej spisze się w aplikacjach, w których na wyjściu użytkownik zainteresowany będzie najbardziej aktualnymi zbiorami częstymi. Natomiast modele Fading Window i Time-tilted Window najlepiej spiszą się, gdy na wyjściu użytkownik zainteresowany jest ostatnimi zbiorami częstymi i jednocześnie kładzie się uwagę na starsze dane, których częstotliwość występowania w przyszłości może ulec zmianie.

Kolejnym elementem pracy było zaimplementowanie algorytmu estDec wykorzystującego model Fading Window do wyszukiwania zbiorów częstych w strumieniach danych.

Jest to algorytm, który może być dobrym dodatkiem do systemów magazynowych lub aplikacji monitorowania sieci. Z założenia jest on łatwy do zrozumienia i prosty do zaimplementowania. Wyniki eksploracji podane przez algorytm dobrze odzwierciedlają zbiory częste pojawiające się w strumieniu danych. Jednocześnie przy pomocy podawanych na początku parametrów  $b$  i  $h$  można w prosty sposób skonfigurować algorytm do własnych potrzeb w zależności czy użytkownik chce zwrócić uwagę tylko na ostatnie zbiory częste w strumieniu bądź preferuje, aby starsze dane zanikały wolniej. Co więcej, podanie odpowiednich parametrów  $S_{min}$ ,  $S_{ins}$  oraz  $S_{prn}$  wpływa na jakość zbiorów częstych w wynikach analizy. Jednak, ze względu na potrzebę generowania zbioru potęgowego dla każdej transakcji, większa ilość elementów w transakcji skutkuje zwiększonym zapotrzebowaniem na pamięć oraz długim czasie przetwarzania.

Reasumując, algorytmy odnajdujące zbiory częste w strumieniu danych wykorzystują dany model do eksploracji w zależności od potrzeb i wymagań dla postawionego problemu badawczego. Algorytm estDec wykorzystuje model Fading Window do odnajdywania ostatnich zbiorów częstych. Ostatnich jest tu słowem kluczowym, ponieważ stare i nieistotne dane nie są brane pod uwagę i zostają usunięte. Dodatkowo, wyniki wygenerowane przez estDec można prosto

zinterpretować. Jednak większa uwaga powinna zostać przykuta do czasu przetwarzania transakcji o dużej ilości elementów.

Zważywszy na ciągle postępujący proces informatyzacji, strumienie będą generowały w przyszłości coraz większe ilości danych. W naturalny sposób zwraca to uwagę na konieczność przeprowadzenia dalszej analizy badawczej przedstawionego w tejże pracy zagadnienia.

## Literatura

- [1] HebaTallah Mohamed Nabil, Ahmed Sharaf Eldin, Mohamed Abd El-Fattah Belal, *Mining Frequent Itemsets from Online Data Streams: Comparative Study*, International Journal of Advanced Computer Science and Applications, Vol. 4, No.7, 2013, s. 117 - 125
- [2] Joong Hyuk Chang, Won Suk Lee, *Finding Recent Frequent Itemsets Adaptively over Online Data Streams*, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 2003, s. 487 – 492
- [3] Stratos Mansalis, Eirini Ntoutsi, Nikos Pelekis, Yannis Theodoridis, *An Evaluation of Data Stream Clustering Algorithms*, Statistical Analysis and Data Mining, 2018 (11, nr 4)
- [4] Cay S. Horstmann, Gray Cornell, *Java. Podstawy. Wydanie IX*, Helion, 2013, s. 30-32
- [5] Urszula Piechota, Jacek Piechota, *JavaFX 9. Tworzenie graficznych interfejsów użytkownika*, Helion, 2018
- [WWW, 6] <https://stackify.com/popular-programming-languages-2018/>, z dnia 08.01.2020
- [WWW, 7] <https://javastart.pl/baza-wiedzy/frameworki/javafx>, z dnia 08.01.2020
- [WWW, 8] <https://en.wikipedia.org/wiki/NetBeans>, z dnia 08.01.2020
- [9] Agrawal R., Imieliński T., Swami A., *Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD international conference on management of data, Washington, DC*, ACM, New York, 1993, s. 207 – 216
- [10] Margara A., Rabl T., *Definition of Data Streams*, Encyclopedia of Big Data Technologies, Springer International Publishing AG, 2018
- [WWW, 11] [http://smurf.mimuw.edu.pl/uczesie/?q=kombinatoryka\\_8](http://smurf.mimuw.edu.pl/uczesie/?q=kombinatoryka_8), z dnia 21.01.2020
- [WWW, 12] [https://en.wikipedia.org/wiki/Spring\\_Framework](https://en.wikipedia.org/wiki/Spring_Framework), z dnia 21.01.2020

## Wykaz ilustracji

<i>Rysunek 1. Model Landmark Window</i> .....	13
<i>Rysunek 2. Model Fading Window</i> .....	14
<i>Rysunek 3. Model Sliding Window</i> .....	15
<i>Rysunek 4. Model Time-tilted Window</i> .....	16
<i>Rysunek 5. Kompromis dokładności, czasu przetwarzania i wykorzystanej pamięci</i> .....	17
<i>Rysunek 6. Metoda estDec</i> .....	24
<i>Rysunek 7. Struktura projektu</i> .....	37
<i>Rysunek 8. Zawartość pakietu com.gurgaczj</i> .....	37
<i>Rysunek 9. Zawartość pakietu com.gurgaczj.algorithm</i> .....	38
<i>Rysunek 10. Zawartość pakietu com.gurgaczj.algorithm</i> .....	38
<i>Rysunek 11. Zawartość pakietu com.gurgaczj.model</i> .....	38
<i>Rysunek 12. Diagram aktywności przetwarzania transakcji</i> .....	40
<i>Rysunek 13. Drzewo trie</i> .....	41
<i>Rysunek 14. Diagram aktywności aktualizowania licznika</i> .....	42
<i>Rysunek 15. Diagram aktywności opóźnionego wstawiania</i> .....	44
<i>Rysunek 16. Diagram aktywności wybierania zbiorów częstych</i> .....	46
<i>Rysunek 17. Diagram aktywności wymuszonego odcinania</i> .....	47
<i>Rysunek 18. Interfejs użytkownika</i> .....	48
<i>Rysunek 19. Dane przygotowane na potrzeby pierwszego testu</i> .....	49
<i>Rysunek 20. Wyniki pierwszego testu</i> .....	50
<i>Rysunek 21. Wyniki drugiego testu</i> .....	51
<i>Rysunek 22. Wyniki trzeciego testu</i> .....	52



## Streszczenie

Odkrywanie zbiorów częstych w strumieniach danych jest formą odnajdywania powtarzającego się wzoru w danym strumieniu. Algorytmy wyszukujące zbiory częste korzystają z różnych modeli w zależności czy uwaga jest zwrócona w stronę wszystkich danych strumienia lub tylko ostatnich.

Głównym celem niniejszej pracy jest przedstawienie modeli do eksploracji danych w strumieniach danych oraz implementacja algorytmu estDec do wyszukiwania ostatnich zbiorów częstych w strumieniach danych. Algorytm ten adoptuje model Fading Window, w celu zwrócenia większej uwagi na nowe dane.

Aplikacja została napisana w języku Java przy pomocy takich bibliotek i frameworków jak Spring, JavaFX oraz Google Guava.

Discovering frequent itemsets in data streams is a form of finding a repeating pattern in a given stream. Algorithms that search for frequent sets use different models depending on whether the attention is directed to all stream data or only the last one.

The main purpose of this paper is to present models for data mining in data streams and to implement the estDec algorithm, which finds recent frequent itemsets in data streams. This algorithm adopts the Fading Window model so that it is possible to pay more attention to new data.

The application was written in Java using libraries and frameworks such as Spring, JavaFX and Google Guava.

## **Załączniki**

### **1. Zawartość płyty CD**

- I. Praca dyplomowa w wersji elektronicznej.
- II. Kod źródłowy aplikacji implementującej algorytm estDec.
- III. Gotowy do uruchomienia plik skompilowanej aplikacji.
- IV. Pliki .csv wykorzystywane do testów aplikacji.